UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Diseño e implementación de un sistema de visualización de datos de fuentes abiertas

Francisco Antonio Vargas Cánovas
Tutor: Álvaro Ortigosa Juárez

JUNIO 2018

Diseño e implementación de un sistema de visualización de datos de fuentes abiertas

AUTOR: Francisco Antonio Vargas Cánovas
TUTOR: Álvaro Ortigosa Juárez

Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio de 2018

Resumen

Vivimos en la era de los datos. En los últimos años, la potencia de proceso ha crecido, el almacenamiento se ha abaratado, las comunicaciones se han vuelto más globales y rápidas que nunca y la cantidad de dispositivos electrónicos conectados ha aumentado exponencialmente. Esto ha resultado en una cantidad de datos enorme, cuantificando todos los aspectos de los negocios y la sociedad. La capacidad de recoger, analizar y mostrar estos datos en un tiempo mínimo influye en la toma de decisiones en todo tipo de sectores, como el bancario, el industrial, el científico o el educativo.

Por esta razón, se ha trabajado en el diseño y desarrollo de una plataforma completa para esta tarea. Desde los módulos que recogen datos de fuentes abiertas al público hasta los que muestran un panel de resultados de los análisis, todos los módulos han sido diseñados pensando en la flexibilidad como meta principal. Como ejemplo se ha utilizado Twitter como fuente abierta, y un analizador de sentimientos para procesar los datos recibidos, que se muestran de forma gráfica, de mapa y de tabla. El tiempo que los desarrolladores han de usar para cambiar de fuente de datos, analizador o panel de resultados debe ser el mínimo posible, por lo que se han utilizado el concepto de virtualización para aislar la ejecución de la plataforma de la configuración y entorno de los desarrolladores, asegurándose que el código funcionará de la misma forma tanto en las máquinas de desarrollo como en las de pruebas y producción.

Para ello, se ha hecho un estudio de las tecnologías usadas en la actualidad, buscando aquellas que permitan la mayor agilidad, portabilidad y escalabilidad. Las herramientas usadas están en constante evolución, muy documentadas y mantenidas por la comunidad del software libre.

Por último, se explicarán las fases del desarrollo y puesta en producción de la plataforma, así como el proceso de modificación de esta para una futura adaptación a otras fuentes o analizadores.

Palabras clave

Análisis de datos, análisis de sentimientos, virtualización, Docker, JavaScript, Node.js

Abstract

We are living in the age of data. In the last years, processing power has increased, storage costs plummeted, communications became more global and faster than ever and the number of connected electronic devices has grown exponentially. This resulted in a huge amount of data, quantifying every aspect in business and society. The capability of fetching, analyzing and showing this data in the shortest time possible affects in decision making in all kinds of professional sectors, like in industry, banking, scientific or educational.

Because of this, work has been done in design and development of a comprehensive platform for this process. Starting from the data fetching modules, which collect data from open sources, to the ones that show a dashboard with the analyzed data, every module has been designed thinking in flexibility as the main goal. For this project, Twitter has been chosen as the source, a sentiment analyzer processes the received data, and a dashboard that shows the processed data in several ways: Geographically, in a chart and in a table. Developers must use the shortest time to swap the data sources, the analyzer or the processed data dashboard, so the concept of virtualization has been used to isolate the platform execution from the developers' setup and environment, making it sure that the code will work the same way in development machines as in testing or production.

For it, a study has been made about currently used technologies, looking for those that allow developers the maximum agility, portability and scalability. The used tools are constantly evolving, properly documented and maintained by open source community.

Finally, platform development and production stages will be explained, as well as the process a development team must follow to adapt the platform to new data sources or analyzers.

Keywords

Data analysis, Sentiment analysis, Virtualization, Docker, JavaScript, Node.js

Agradecimientos

A mi tutor, Álvaro, por haber confiado en mí y haberme guiado por este proyecto con simpatía y respeto, y por ofrecerme recursos software (¡y hardware!) para llevarlo a cabo.

A Diego y Fernando, porque después de todo, he aprendido más de ellos que ellos de mí.

A mi familia, por el apoyo recibido todos estos años.

A Elena, por ser el CSS de mi HTML.

Y por supuesto, a toda la comunidad de desarrolladores de código abierto, porque este trabajo se habría vuelto mucho más teórico y/o caro realizarlo sin su esfuerzo.

"If you torture the data long enough, it will confess to anything."

Ronald Coase, economista británico.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

1.1 Motivación

Este trabajo busca dar un nuevo enfoque al desarrollo de aplicaciones, utilizando herramientas y paradigmas que permitan a los desarrolladores trabajar de una forma más sencilla y eficiente.

Uno de los problemas con el que se enfrentan los desarrolladores es encontrar a usuarios u otros desarrolladores que tienen problemas para utilizar sus programas porque estos fallan en sus sistemas. "En mi ordenador funciona" [1] es una frase común en los equipos de desarrollo de todo el mundo. Este problema tiene su raíz en los diferentes entornos y configuraciones que desarrolladores, ingenieros de pruebas, administradores de servidor y usuarios finales utilizan para crear, probar y ejecutar sus programas. En este TFG se exponen diferentes formas de atajar este problema, basándose en el concepto de virtualización.

Por otro lado, se parte de una herramienta ya creada, capaz de analizar las emociones del autor a partir de sus textos [2], para adaptarla e integrarla en una plataforma completa que recoge información, la envía al analizador y muestra los resultados en un panel. Si bien hay diferentes maneras de realizar esta plataforma, en este TFG se propone una que aporta flexibilidad y escalabilidad, usando herramientas de software libre y ampliamente documentadas.

1.2 Objetivos

El objetivo de este trabajo es crear una plataforma completa de visualización de datos, partiendo desde la recogida de estos de fuentes de acceso público, su análisis, almacenamiento y el servicio de consulta por parte de los usuarios, siempre tomando en cuenta el requisito de permitir diferentes analizadores y fuentes de datos.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- Estado del arte: Un resumen sobre las tecnologías que se deben utilizar para llevar a cabo el proyecto, valorando las diferentes opciones de uso y explicando textual y gráficamente su funcionamiento.
- Diseño: En este capítulo se tratará lo que debe hacer la herramienta, y cómo se propone conseguirlo en la fase de desarrollo, atendiendo a lo explicado en el Estado del arte.
- Desarrollo: Una vez diseñada la aplicación, se explica paso a paso cómo se ha llevado a cabo, teniendo en cuenta aspectos importantes a tomar en cuenta
- Pruebas y calidad del código: Se explica cómo se ha probado el correcto funcionamiento de la herramienta, así como los módulos utilizados para llevar a cabo las pruebas y mantener un nivel de calidad en el código.
- Conclusiones y trabajo futuro: Se resume el trabajo realizado en este proyecto, y se proponen algunas mejoras al mismo.

2 Estado del arte

2.1 Tecnología de servidor

Tradicionalmente, una máquina física ha alojado un sistema operativo, sobre el que estaban instalados las aplicaciones y librerías necesarias para su funcionamiento. En el caso de un servidor configurado para su uso como aplicación web, este debía tener instalado un servidor HTTP que recibiera las peticiones de los clientes, un servidor de base de datos y un motor de aplicación que permitiera generar el código HTML que finalmente recibía el usuario. El conjunto más conocido de aplicaciones para montar esta infraestructura es conocido por el acrónimo LAMP: Linux como sistema operativo, Apache como servidor Web, MySQL como servidor de bases de datos y PHP, aunque también Python o Perl, como lenguaje de programación para generar el código HTML final. [2]

Esta infraestructura tiene varias desventajas. Si el sistema ejecuta más de una aplicación, se debe tener cuidado para que la primera aplicación no entre en conflicto con la segunda y sucesivas.

Por otro lado, replicar la configuración exacta del servidor de producción en equipos de desarrollo o pruebas puede no ser posible, en cuyo caso es muy probable encontrar un código que funciona en la configuración de desarrollo, pero no en pruebas, o un funcionamiento diferente en desarrollo y producción.

Por último, escalar la infraestructura es costoso tanto en recursos humanos como económicos. Al no poder ajustar los recursos a medida de la carga en el servidor, es muy posible que la mayor parte del tiempo se estén malgastando recursos, o por el otro lado, haya picos de carga que el servidor no sea capaz de procesar. La única forma de reajustar los recursos sin preocuparse de juntar varias aplicaciones para evitar conflictos es cambiando el hardware y reconfigurando las aplicaciones.

2.1.1 Virtualización

Para solucionar estos problemas se acude a la virtualización. La virtualización crea una capa de abstracción entre el servidor físico y las aplicaciones, creando máquinas virtuales, donde el sistema operativo instalado en ellas está aislado del resto de máquinas virtuales, así como de la máquina física. De este modo, cada máquina puede tener su configuración, aplicaciones y dependencias sin preocuparse por los conflictos que pueda generar con otras aplicaciones del sistema. [3]

Figura 2-1: Diagrama de arquitectura de sistema virtualizado.

En esta arquitectura, es el monitor de hardware virtual o *hypervisor* el que se encarga de asignar recursos físicos a las máquinas virtuales (como núcleos de CPU, memoria RAM o de vídeo y dispositivos USB) que las máquinas virtuales reconocerán como propios. Por otro lado, es capaz de emular redes de comunicaciones y dispositivos de almacenamiento. Por ejemplo, lo que para una máquina virtual es un disco duro, para el *hypervisor* es un archivo. Esta característica hace que realizar copias de seguridad del sistema sea tan sencillo como copiar un archivo, en vez de hacerlo a través de imágenes de disco duro.

Además, se pueden realizar instantáneas o *snapshots* de las máquinas virtuales, pudiéndose clonar y ejecutar en otras máquinas físicas (siempre que el *hypervisor* sea compatible), haciendo el proceso de migración mucho más sencillo, así como acabar con gran parte de los problemas de incompatibilidad entre sistemas, ya que cada equipo de desarrollo puede tener una instancia de la máquina virtual, seguros de que su código funcionará de la misma forma en las máquinas de pruebas y de producción.

En los últimos años, la tecnología de virtualización ha ido mejorando lo suficiente como para que su uso no suponga una fuerte reducción en el rendimiento de los sistemas ejecutados. Las mejoras en los procesadores y en los *hypervisors* han reducido la diferencia de rendimiento entre servidor físico y virtual a menos de un 10%, algo perfectamente asumible si se tienen en cuenta todas las ventajas que proporciona este paradigma. [4]

Sin embargo, si ha revolucionado la industria ha sido por su impacto económico. No sólo porque una empresa puede consolidar sus servidores en menos máquinas, o reajustar los recursos asignados a una instancia únicamente con un reinicio de esta. [5] Se ha creado todo un mercado de servidores virtuales privados o VPS, con el que una empresa de alojamiento puede vender recursos de sus máquinas físicas para crear máquinas virtuales, a las que el cliente podrá conectarse remotamente a través de una IP de entrada asignada a la máquina y configurar del mismo modo que haría con un servidor físico, gracias a la abstracción del hardware. Actualmente, empresas del sector del hosting como OVH ofrecen servidores virtuales por 4€ al mes, asignando 1 núcleo de CPU, 2 GB de memoria RAM y 20 GB de espacio en disco [6], algo más que suficiente para proyectos personales y aplicaciones para pequeñas y medianas empresas. En el ámbito de servidores físicos ninguna empresa ofrecerá una máquina tan limitada, por lo que la opción más reducida sería mucho más costosa y se derrocharían muchísimos recursos (4 núcleos de CPU, 32 GB de RAM y 4 TB de disco duro por 59,99 €/mes). [7]

Además, en los últimos años, empresas como Amazon, Digital Ocean o Alibaba han popularizado las instancias de máquina virtual bajo demanda. En vez de facturarse mensual o anualmente y elegir de antemano los recursos asignados, el cliente, según le sea necesario, puede crear máquinas virtuales en con diferentes recursos en un tiempo medio de un minuto, destruirlas cuando dejen de ser necesarias y/o modificar los recursos de estas, siendo facturado por hora en función de la capacidad de las instancias creadas. Las empresas que ofrecen este servicio tienen además una API que permite la gestión de las instancias de forma automática. Dos posibles casos de uso son: crear una máquina para alojar una aplicación que debe funcionar durante un corto periodo de tiempo (por ejemplo, una simulación científica que requiera mucha potencia o computación usando GPU, o una aplicación web que deba prestar servicio durante unas jornadas), poder ajustar los recursos (CPU y memoria RAM) a demanda para ajustarlo a la carga en función del momento (por ejemplo, si a través de la monitorización se halla un patrón en el que los usuarios de un servicio se conectan más a

determinadas horas, pueden asignarse más recursos en las horas pico y liberarlos en las horas valle, facturando por la capacidad necesaria).

Para poner en comparación con los otros dos servicios, en Digital Ocean, una instancia virtual con capacidad similar al VPS antes mencionado (2 GB RAM, 1 núcleo CPU, 50 GB disco SSD) cuesta 10$ al mes [8] (al cambio a fecha de este trabajo, 8.60 EUR). Sin embargo, al poder facturarse de forma horaria, cada hora se pagaría a 0.015 $ (0.013 EUR). Por tanto, si no se desea modificar los recursos de la máquina, un VPS tradicional es una opción más rentable; pero si no se piensa contratar a largo plazo, o si se pretende reajustar recursos o crear y destruir instancias a medida de la necesidad del cliente, las instancias bajo demanda resultan mucho más atractivas.

Se debe dejar claro que esta virtualización se realiza a nivel de hardware, asignando recursos de la máquina física a las máquinas virtuales. Cada máquina virtual tiene un sistema operativo instalado, independiente del sistema operativo de la máquina física.

2.1.2 Docker

Por otro lado, existe la virtualización a nivel de sistema operativo. También llamada contenerización.

En la virtualización a nivel de sistema operativo el núcleo o *kernel* aísla la ejecución del servicio iniciado del resto del sistema. La principal restricción de este sistema es que tanto el host como la instancia deben utilizar el mismo núcleo, por lo que no se permitiría la utilización de sistemas operativos diferentes (o incluso versiones del *kernel* diferentes), como sí ocurre con la virtualización a nivel de hardware. Este paradigma elimina la capa del sistema operativo en la virtualización, lo que se traduce en instancias más livianas, tiempos de creación más rápidos y un rendimiento similar a los sistemas ejecutados directamente contra la máquina física. [9]

De esta forma, las librerías y aplicaciones necesarias para el funcionamiento de la aplicación están aisladas del host en su propio contenedor, pero usando directamente en el *kernel* del sistema operativo de la máquina.

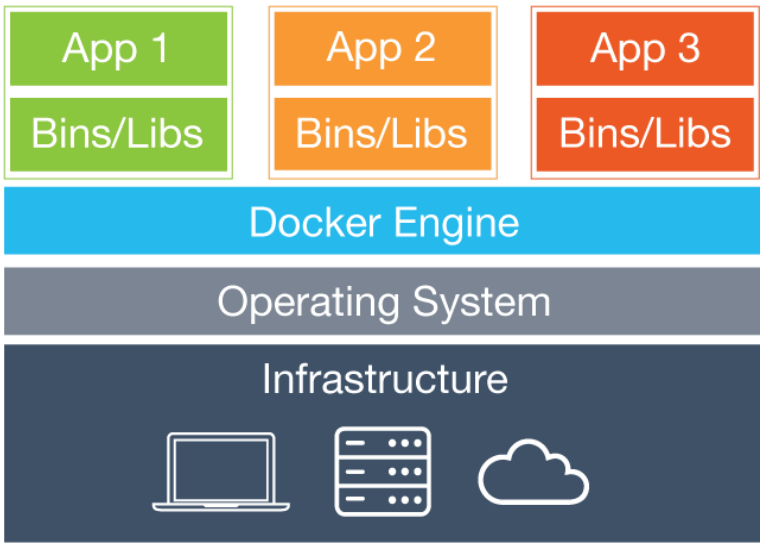

Figura 2-2: Diagrama de arquitectura de contenedores.

Si bien la virtualización a nivel de sistema operativo ha sido usada desde hace más de 15 años con Jails en FreeBSD o LXC o Linux Containers, ha sido Docker quien ha conseguido popularizarlo en la industria. [10]

Docker permite a los desarrolladores crear contenedores que son reutilizables en diferentes máquinas. [11] A partir de una imagen inicial se van añadiendo cambios (instalación de librerías, aplicaciones, configuraciones, etc.) que permiten crear una imagen final, de la que se pueden generar contenedores que se ejecutan en el sistema. Docker se encargará de conectar el contenedor con el *kernel*, y de proveer una red de comunicaciones y un sistema de ficheros al contenedor, aislándolo del resto de contenedores y de la máquina host. La creación de imágenes se puede realizar a través de un fichero de configuración o Dockerfile [13], que utiliza un DSL o Lenguaje específico de dominio para automatizar la generación de la imagen. Los comandos van desde la ejecución de comandos en el contenedor a la inclusión de archivos de la máquina host para su uso en el contenedor.

Por otro lado, Docker suministra una serie de herramientas que permiten al usuario ejecutar comandos en el contenedor una vez ha sido creado, enlazar puertos de la máquina física con el contenedor o conectar directorios del sistema host con el contenedor, de tal modo que ambos puedan acceder a los mismos archivos, o si se trata de un contenedor que funciona como servidor, poder conectarse como cliente desde fuera del contenedor o de la propia máquina física.

Otro gran pilar de Docker es que tanto la industria como la comunidad de software libre se están apoyando mucho en esta tecnología, permitiendo a los desarrolladores utilizar sus proyectos dentro de Docker de una forma muy sencilla. Como ejemplo, la puesta en marcha de un blog basado en Wordpress de la forma tradicional requiere la instalación y configuración de un servidor web, un servidor de bases de datos MySQL, la conexión a un motor de script PHP (la pila LAMP explicada anteriormente), configurar Wordpress para su uso en el servidor y por último configurar el blog. Además de requerir los conocimientos necesarios, es necesario invertir tiempo en esta tarea, y se corre el riesgo de alterar la configuración de las aplicaciones ya funcionando en el servidor. Con Docker, el desarrollador podría descargar la imagen oficial de Wordpress, la imagen oficial de MySQL server, iniciar un contenedor de cada una, además de enlazar el puerto 80 a un puerto de la máquina host. Dependiendo de la velocidad de la red, en menos de un minuto se podrá acceder al contenedor recién creado a través de un navegador para configurar el blog, ya que toda la configuración previa e instalación de librerías y aplicaciones necesarias ya fue realizada por otros desarrolladores, que publicaron la imagen final en Internet para su descarga. Docker, a través de su sitio Docker Store [12], permite la publicación de imágenes creadas por los desarrolladores, donde estos pueden mostrar una página con documentación necesaria para su puesta en marcha y funcionamiento. El motor Docker instalado en el sistema buscará en este repositorio la imagen que el desarrollador desea utilizar, importándola si fuera necesario.

En la gestión de arquitecturas basadas en contenedores, Docker incluye herramientas como Docker Compose, que permite la creación de una red de contenedores a partir de un archivo de configuración, y su gestión conjunta posterior en la máquina host, y Docker Swarm, que permite la gestión de varios motores Docker instalados en diferentes máquinas, pudiendo desplegar contenedores y servicios en diferentes servidores a la vez, así como gestionarlos y

monitorizarlos remotamente. Docker es compatible herramientas como Kubernetes, desarrollada por Google, que permite automatizar tareas relacionadas con contenedores.

Todas estas ventajas han hecho que la industria comience a adoptar Docker en sus estrategias, llegando a un 25% en tan solo 4 años. [13]

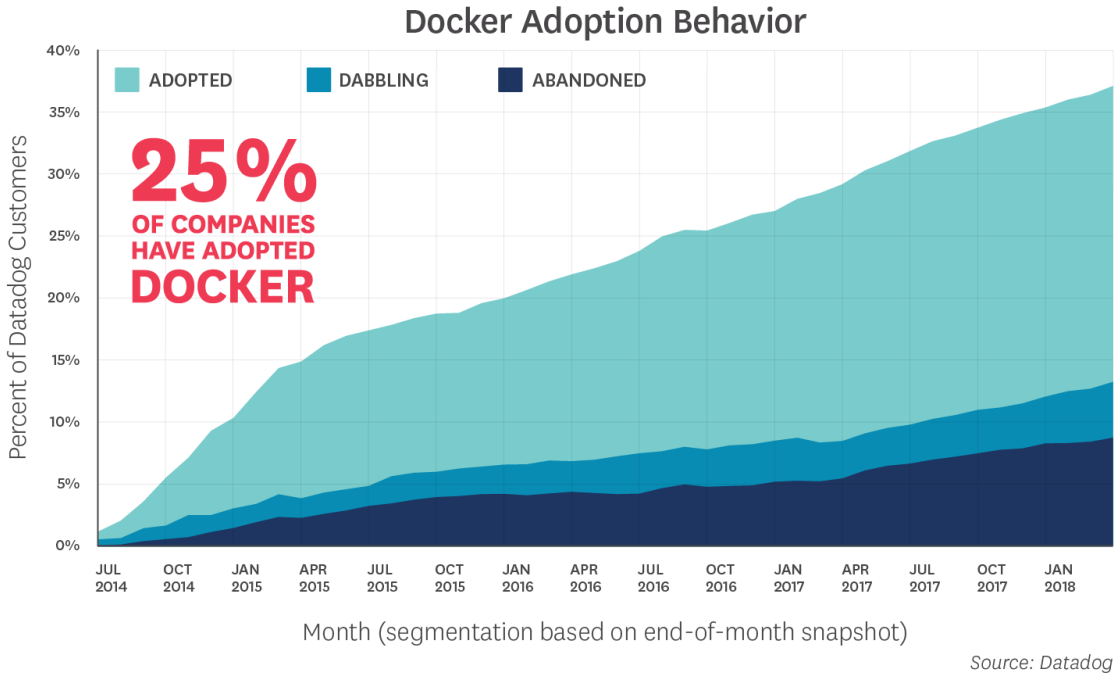

Figura 2-3: Adopción de Docker por parte de empresas en los últimos 4 años [13].

2.2 Tecnologías de desarrollo web

2.2.1 JavaScript y Node.js

El lenguaje JavaScript, inicialmente concebido para su uso en web para crear interactividad en páginas web a partir de código ejecutado en el lado del cliente, es decir, dentro del propio navegador, dio el salto al lado de servidor gracias a Node.js [14], que en el motor V8 de JavaScript usado por Google en su navegador Chrome para ejecutar código JavaScript en aplicaciones.

Igual que el motor JavaScript en un navegador espera a que ocurran eventos para ejecutar un bloque de código (por ejemplo, un clic del ratón o una pulsación en el teclado), Node.js puede escuchar y actuar ante eventos de interfaces de entrada y salida, como el sistema de archivos, bases de datos, red u otros dispositivos.

Por otro lado, Node.js utiliza un sólo hilo de proceso, aprovechando el uso de funciones asíncronas para acceder a las interfaces de entrada y salida. Este sistema, en vez de crear un hilo para cada petición de un recurso, como haría un servidor tradicional, los añade a una cola de eventos, que quedan en espera hasta que la interfaz que puede proveer su recurso queda disponible. Una vez sucede esto, la petición se ejecuta, llamando a una función definida en la solicitud con el recurso como parámetro. Esta función se conoce como *callback*. Esta arquitectura no bloquea la ejecución de la aplicación hasta que el recurso es recibido, ya que el sistema estará atendiendo otras peticiones hasta que pueda ejecutar la

primera petición. Si bien esto implica una forma diferente de escribir el código de una aplicación, su uso correcto permite tasas de conexiones concurrentes mucho mayores que en servidores tradicionales. Como ejemplo, empresas como Netflix, Uber o Paypal usan Node.js en sus servidores en producción. [15]

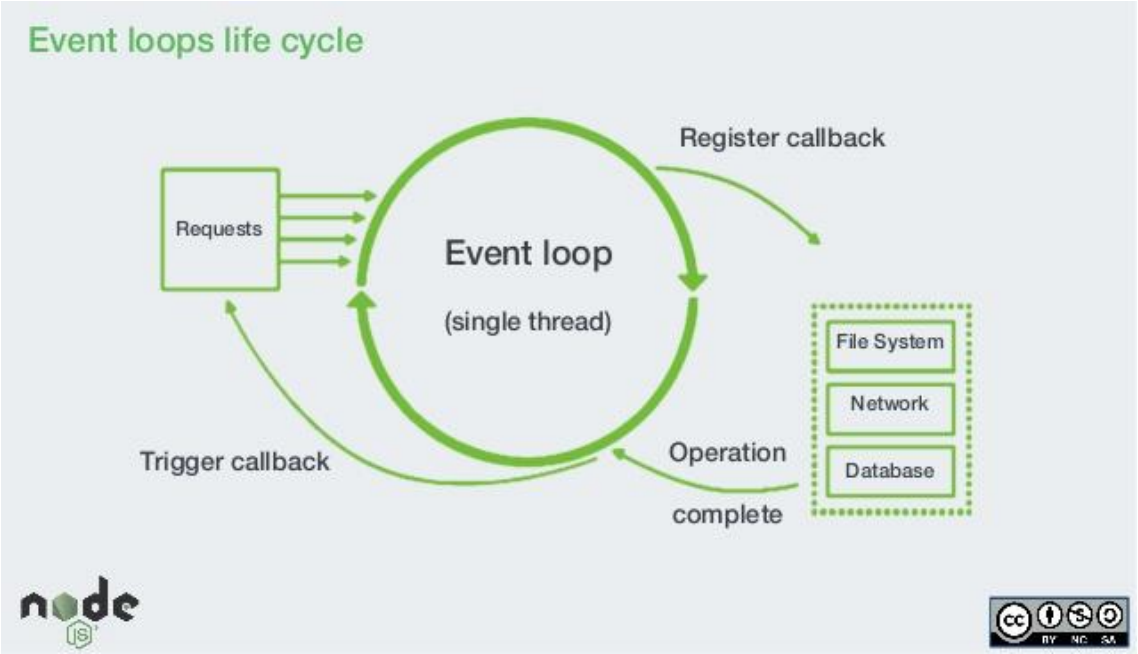

Figura 2-4: Diagrama de funcionamiento basado en eventos de Node.js.

Otro de los valores añadidos de usar JavaScript como lenguaje en el lado de servidor es que permite que un desarrollador use el mismo lenguaje tanto en *back-end* (parte del software que conecta con los datos y se ejecuta en el servidor) como en el *front-end* (parte que se encarga de la visualización de los datos, y dependiendo de la tecnología se ejecuta en cliente o servidor).

La comunidad de profesionales y aficionados del desarrollo StackOverflow recogió en su encuesta anual [16] que un 57.9% de los usuarios se identificaba como "*Desarrollador Back-end*", un 37.8% como "*Desarrollador Front-end*" y un 48.2% como "*Desarrolladores Full-Stack*" (desarrolladores que trabajan en ambas partes del código). En el apartado de lenguajes utilizados, un 69.8% declaró escribir código JavaScript, y un 49,6% utilizar Node.js. Porcentajes casi idénticos se muestran si se filtra sólo a desarrolladores profesionales. Poder utilizar JavaScript en ambos lados de una aplicación ha beneficiado también a estos desarrolladores.

En cuanto al lado del cliente, la librería jQuery, que permite un uso más sencillo de JavaScript para manejar el DOM, sigue dominando el mercado (se estima su uso en 70 millones de sitios web a fecha de este trabajo). [17] Su facilidad de uso, pequeño tamaño y que es compatible con la práctica totalidad de navegadores en el mercado (dado que cada navegador implementa su propio motor JavaScript, no se puede asegurar que la ejecución de un script sea idéntica en todos ellos) han hecho que su conocimiento se considere casi un requisito a la desarrollar código de front-end.

Sin embargo, en los últimos años las revisiones del lenguaje JavaScript han incluido funciones en su código base que permiten la manipulación del DOM de una forma muy similar a la que permite jQuery. Además, la caída en el uso del navegador Microsoft Internet Explorer ha permitido que muchos desarrolladores opten por no dar compatibilidad a este navegador, evitando muchos problemas de compatibilidad entre motores JavaScript. [18]

Además, hay que destacar la proliferación del uso de *frameworks* para front-end basados en JavaScript. Si bien han incrementado la complejidad del desarrollo, permiten la reutilización y modularidad del código, partiendo los diferentes bloques de la capa de presentación en componentes que se conectan entre sí. Además, permiten independencia del back-end, ya que los datos los recibirá a través de HTTP. Los más usados por la comunidad son Angular, creado por Google, y React, creado por Facebook. Recientemente el *framework* Vue está ganando popularidad debido a su curva de aprendizaje más corta, si bien aún no alcanza a los otros dos antes mencionados. [19]

2.2.1.1 NPM

Por último, no se puede hablar de Node.js y JavaScript sin mencionar a NPM. Igual que uno de los puntos fuertes de Docker era la facilidad con la que los desarrolladores podían publicar sus imágenes para el uso de la comunidad, el gestor de paquetes de Node.js permite a los desarrolladores usar más de 600.000 paquetes publicados (a fecha de este trabajo) [20] que se pueden descargar e importar en el código de un programa para Node.js. Para facilitar su uso, en cada proyecto de Node.js se puede incluir un archivo **package.json** con la lista de dependencias que deben incluirse en la aplicación para que el código pueda ejecutarse, siendo lo único que es necesario incluir con el código a la hora de publicarlo en un repositorio, ya que la descarga de dependencias se realizará de forma independiente descargando estas a través de NPM en cada equipo. Este sistema es similar al utilizado por PHP con Composer, Python con Pip, .Net con NuGet o Java con Maven.

Todas estas novedades en el mundo del desarrollo web han popularizado la pila MEAN, acrónimo para M de MongoDB, una base de datos no relacional, E de Express, un *framework* web para código de servidor en Node.js, Angular como *framework* web de código en el lado de cliente y Node.js como motor de estos dos últimos. Además, añadiendo componentes como Apache Cordova o Electron se pueden convertir aplicaciones web en aplicaciones móviles o de escritorio respectivamente, sin modificar gran parte del código ya realizado y sin utilizar el lenguaje nativo de las plataformas donde se acabarán usando estas aplicaciones. [21][22]

3 Análisis y diseño

3.1 Análisis

El problema a solucionar radica en la necesidad de mostrar los datos que produce el analizador. Actualmente este analizador funciona de manera independiente, ejecutándose como un programa en el que dentro de su código se especifica el texto a analizar. Esta forma de utilizar la herramienta no permite el análisis de datos en tiempo real, por lo que debe adaptarse.

Por tanto, los requisitos que debe reunir la herramienta para su uso completo son:

- Requisito funcional 1 (RF1): El analizador debe recibir peticiones externas con datos en bruto, que serán analizados por este, y devolver el resultado al proceso solicitante.
- RF2: La herramienta debe recoger datos de diferentes fuentes, pudiendo modificar el origen de los mismos. Estos datos pueden ser de diferente naturaleza.
- RF3: La herramienta debe permitir el uso de diferentes analizadores, que pueden devolver resultados de diferente tipo en función de los análisis que realicen.
- RF4: La herramienta debe proveer un modo de visualizar los datos procesados
- RF5: La herramienta debe permitir el uso de añadir diferentes modos de visualización, a partir de módulos que puedan recibir los datos procesados por el analizador.
- RF6: La herramienta debe permitir que alguno de los módulos que la componen se encuentren en un servidor externo, o no esté desarrollada por el equipo de desarrollo.
- RF7: La herramienta debe almacenar los datos en bruto recogidos, así como los datos procesados por el o los analizadores utilizados.

Además, se especifican los siguientes requisitos no funcionales:

- Requisito no funcional 1 (RNF 1): La herramienta debe funcionar tanto en sistemas operativos basados en UNIX como en los basados en Microsoft Windows.
- RNF 2: La herramienta debe estar desarrollada utilizando herramientas de código abierto.
- RNF 3: La herramienta debe tener una arquitectura modular, que permita la conexión de nuevos módulos y la comunicación entre ellos para realizar las funciones.

3.2 Diseño

Para diseñar la plataforma, se ha tenido en cuenta un aspecto muy importante: la flexibilidad. Si bien el desarrollo se ha realizado sobre un posible conjunto de entradas, procesos y salidas, la plataforma debe estar preparada que recoger datos de otras fuentes, analizarlos de diferente forma y mostrarlos o enviarlos al usuario sea lo más fácil posible, sin necesidad de transformar la plataforma entera.

Por esta razón, se ha dividido la plataforma en diferentes módulos:

3.2.1 Agente de recepción de datos en bruto

Esta parte de la plataforma se conectará con la fuente, que en esta versión es la red social Twitter, para recibir los datos a analizar. Este agente no se debe encargar de manipular los

datos a analizar, salvo el añadido de metadatos para su posterior tratamiento. Este módulo enviará los datos recibidos a su agente de carga.

3.2.2 Agente de carga de datos en bruto

Para aislar el módulo de recogida de datos del resto del sistema, este se conectará únicamente con su agente de carga, que sí tiene conexión con la base de datos, únicamente para escribir en ella. De esta forma, si se decidiera cambiar la base de datos, no sería necesario aplicar cambio alguno en el módulo de recogida de datos. Este módulo recibirá los datos brutos junto con metadatos de tratamiento y los almacenará en la base de datos. Por tanto, este módulo necesita ser desarrollado en función de los datos recibidos y del tipo de servidor de base de datos.

3.2.3 Servidor de base de datos

Almacenará los datos que recibe de los agentes de carga de datos brutos. Asimismo, también se encargará de almacenar los datos procesados por los analizadores. Por la naturaleza dinámica y flexible de los datos, se ha decidido utilizar un servidor de base de datos NoSQL, que permite almacenar datos sin una estructura fija, a diferencia de las bases de datos SQL donde se debe definir una tabla y sus campos, cada uno con un tipo de dato predefinido.

3.2.4 Agente de gestión de datos para el analizador

Siguiendo el mismo razonamiento que en el agente de carga de datos brutos, la base de datos y el analizador estarán comunicados a través de este agente, y no directamente, para permitir mayor libertad a la hora de elegir base de datos sin modificar el código del analizador.

Este módulo realizará la descarga de los datos brutos que el analizador necesita, pudiendo elegir la fuente de estos, especificada en los metadatos escritos por el agente de recepción de datos. Una vez recibidos, puede transformarlos de forma que se adecúen al formato necesario por el analizador. Posteriormente, los enviará al analizador, que responderá con los datos analizados. Por último, se encargará de cargar los datos analizados en la base de datos, marcando los datos brutos de tal modo que no vuelvan a ser analizados de nuevo.

3.2.5 Analizador de datos

En este caso, se utilizará el analizador de sentimientos a partir de textos en español mencionado anteriormente [23] adaptándolo a una arquitectura cliente-servidor. Este analizador recibe texto en español y devuelve el número de palabras analizadas y cuántas de esas palabras en se han reconocido como alegres, tristes, que demuestran enfado o miedo. No es objetivo de este trabajo mejorar el funcionamiento del analizador, simplemente adaptarlo en la plataforma.

3.2.6 Servidor API

Módulo que consultará a la base de datos para recibir los datos procesados, y responde con estos a los clientes que han realizado las peticiones. Este módulo permite que los clientes no necesiten conocer la estructura interna de la base de datos, pues lo único necesario para realizar la petición será una URL y la estructura de los datos de respuesta.

3.2.7 Cliente web

Módulo que conecta con el servidor API y solicita los datos procesados, cuya respuesta alimenta las vistas que finalmente se muestran al usuario. En este caso se desarrollará un cliente web que muestra los datos en forma de mapa geográfico, un mapa de radar y una

tabla. La flexibilidad de la plataforma deberá permitir que se puedan desarrollar nuevos clientes para mostrar los datos de diferente manera, sea web o aplicación de escritorio o móvil.

Adicionalmente, por seguridad y control, se crearán los siguientes módulos:

3.2.8 Agente de copia de seguridad de datos:

Módulo que realizará una copia de seguridad de la base de datos de forma periódica. Este módulo solo necesita conexión a la base de datos, siendo independiente del resto de la plataforma.

3.2.9 Panel de control de estado

Módulo que recibirá una señal periódica generada en cada agente, de tal forma que se tenga la información de la última vez que se recibió dicha señal para poder detectar errores en cualquiera de los módulos controlados. Mostrará la información a través de una web.

Para facilitar las labores de desarrollo, migración, puesta en marcha y mejora de la plataforma, se ha diseñado una arquitectura de microservicios donde cada módulo anteriormente descrito se ejecuta dentro de un contenedor Docker. Para ello, será necesario crear una imagen Docker de cada módulo, y en este caso en el que se dispone de un servidor para desplegar la plataforma, describir en un fichero de configuración de Docker Compose 'docker-compose.yml' [24] la red de contenedores, las dependencias entre ellos y los parámetros que necesitan para funcionar con la configuración deseada. Cada módulo se conectará con otros a través de peticiones HTTP.

Para generar la imagen Docker, cada módulo requiere del código necesario para funcionar y un archivo Dockerfile que permita generar dicha imagen.

Debido a su simplicidad y su facilidad para crear aplicaciones ligeras en el lado de servidor, se propone utilizar Node.js para los agentes de recepción, gestión de base de datos y servidor API.

El analizador, desarrollado en Python, incluirá un módulo, también en Python, que lo adapta a su uso en arquitectura cliente-servidor. Este módulo se construirá usando la librería Flask [24], que permite escuchar peticiones HTTP.

Para desarrollar el cliente web que mostrará los datos procesados, se ha utilizado el *framework* Vue, usando la herramienta 'vue-cli' [25] para generar un *scaffold* o estructura inicial sobre la que se construyen los componentes de la web.

El agente de copia de seguridad de datos se implementará con un script en SH que ejecutará el programa de volcado de datos del servidor MongoDB 'mongodump', y convertirá su salida en un archivo comprimido debidamente nombrado con una marca de tiempo.

Por último, los módulos de cliente y servidor de Panel de control se implementarán en Laravel, un *framework* web que usa el lenguaje PHP [26] para el *back-end*, y Vue para el front-end. Si bien se podría utilizar Node.js como en el resto de agentes, se ha propuesto esta tecnología para demostrar la flexibilidad de la arquitectura, que permite diferentes tecnologías y *frameworks* funcionando simultáneamente sin problemas de conflictos.

En cuanto al hardware, si bien una arquitectura basada en contenedores hace que las características exactas del servidor físico tengan menos importancia a la hora del despliegue, siempre que tenga potencia suficiente, es recomendable tenerlo en cuenta para optimizar la estabilidad y seguridad de la plataforma. Se trata de un servidor ubicado en la UAM, con 16 núcleos Intel Xeon y 32 GB de memoria RAM. Además, contiene 4 discos duros: uno de ellos en estado sólido y tres de tipo tradicional, estos en modo RAID 5 para mayor seguridad.

Se configurará la plataforma ubicando los datos en el conjunto de discos en RAID y el sistema en el disco en estado sólido, ganando velocidad y al mismo tiempo almacenando los datos en un conjunto de discos resistente a fallos.

A continuación, se muestra un diagrama del diseño propuesto:

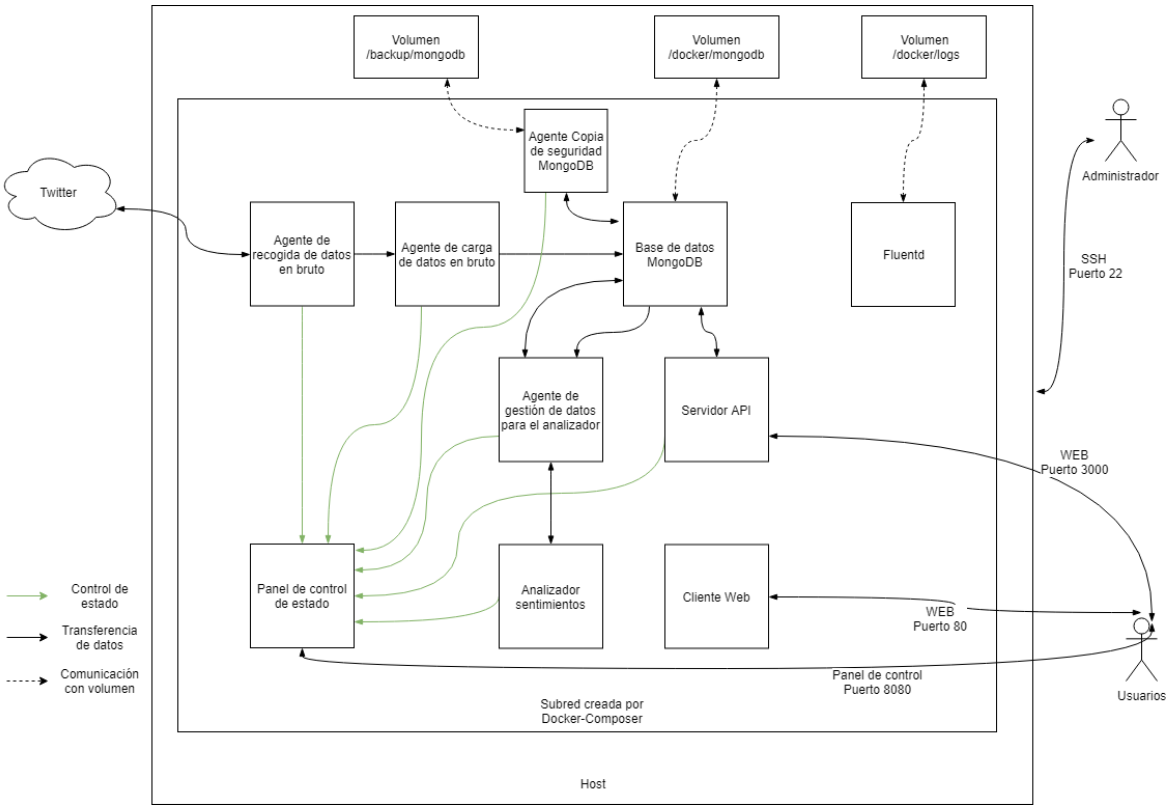

Figura 3-1 Diagrama de arquitectura de contenedores propuesta.

Y, por último, un resumen de los módulos y las tecnologías a utilizar:

Agente de recogida de datos en bruto	Node.js / JavaScript
Agente de carga de datos en bruto	Node.js / JavaScript
Base de datos	MongoDB
Agente de gestión de datos para el analizador	Node.js / JavaScript
Analizador de datos	Flask / Python
Servidor API	Node.js / JavaScript
Cliente web	Vue / JavaScript
Agente de copia de seguridad de base de datos	Script SH
Panel de control de estado	Laravel / PHP

Tabla 1 Resumen de tecnologías propuestas en el diseño para cada módulo.

4 Desarrollo

Dado que se va a utilizar una arquitectura basada en contenedores Docker, hay que tener en cuenta algunos aspectos a la hora de desplegar nuestra plataforma en un servidor:

- Un contenedor debe ser usado para un sólo propósito. Es decir, utilizar un solo contenedor para alojar en él dos servidores va en contra de la buena práctica en arquitectura de microservicios, siendo lo óptimo utilizar un contenedor para cada servidor y enlazarlos entre sí. [27]

- Un contenedor debe alojar la aplicación, no los datos. Para alojar los datos Docker recomienda utilizar volúmenes, y enlazarlos al contenedor o contenedores a través de enlaces a directorios internos. De este modo, el contenedor puede ser detenido, borrado, actualizado o cambiado por otro, pero los datos seguirán estando almacenados dentro del volumen.

Docker tiene dos formas de gestionar los volúmenes: Interna o enlazando a un directorio en el host. De la primera, Docker ubicará los datos en un directorio interno, junto con los contenedores e imágenes. En la mayoría de las situaciones esto es suficiente, pues no es tan importante dónde están alojados exactamente los datos, sino que estos están fuera del contenedor y se pueden usar en diferentes contenedores. [28]

Es la segunda forma la que se usará en este proyecto, pues por la estructura de discos duros de este servidor sí es provechoso especificar la ubicación exacta en el host de los datos.
Para ello, en la configuración del sistema host, se han definido tres particiones:

Punto de montaje	Disco	Capacidad	Observaciones
/ (raíz)	*SSD*	78 GB	Contiene el sistema operativo host, el motor Docker y las imágenes y contenedores.
/docker	*RAID*	1.5 TB	Contiene los directorios que se usarán como volúmenes en los contenedores
/backup	*RAID*	450 GB	Contiene las copias de seguridad de la base de datos, clasificadas por fecha y comprimidas en XZ.

Tabla 2 Puntos de montaje creados en el servidor para su uso en la plataforma.

Para gestionar los archivos de registro, ya que cada módulo informará de su funcionamiento, se podría usar el comando ‘docker-compose logs’, que muestra por pantalla los mensajes diferenciados por contenedor. Sin embargo, se ha decidido usar un servidor Fluentd [29], ya que Docker lo soporta de forma nativa. Fluentd permite configurar el servidor para transformar y almacenar de diferentes formas los registros de las aplicaciones que están conectadas a él. Cada contenedor tiene asociado una etiqueta que lo identifica, y Docker envía a Fluentd los mensajes.

La configuración usada hace que Fluentd recoja los mensajes y los agrupe por contenedor y fecha, rotándolos diariamente y almacenándolos en la carpeta /docker/logs de la máquina host. Para ejecutar este servidor, se usa una imagen Docker del servidor Fluentd [30], que

puede acceder a la carpeta del host antes mencionada, así como al archivo de configuración contenido en la carpeta "fluentd" del código de la plataforma.
Como Docker necesita acceder a este servidor, en el archivo de configuración Docker-compose.yml se ha abierto un enlace al puerto que usa Fluentd para comunicarse, 24224, para que la máquina host lo utilice como propio. Por tanto, cuando el sistema llama a 127.0.0.1:24224, está comunicándose con el servidor Fluentd ejecutado en el contenedor.

Como aspecto común a las imágenes creadas, en la página de referencia de la imagen (por ejemplo, en la de Node.js) [31] se enumeran las diferentes versiones que se pueden utilizar para crear un contenedor. Además de la versión de node.js, que debe ser compatible con nuestro módulo, existen variantes basadas en un sistema Debian y en un sistema Alpine. Alpine [32] es un proyecto que empaqueta las funciones básicas de Linux en un contenedor de 4 MB, mientras que Debian supera los 100MB. Si no se necesitan dependencias o librerías que excedan la capacidad de Alpine, es muy recomendable utilizar esta variante, ya que el tamaño de la imagen, y, por tanto, el contenedor final se reducirá drásticamente. Esta variante también se puede encontrar en otras imágenes, como en Python, que también se usa en esta plataforma.

4.1 Desarrollo de los módulos

4.1.1 Agente de recogida de datos de Twitter:

Para desarrollar este módulo, se ha usado la API de búsqueda de tweets que provee Twitter [33]. A través de peticiones HTTP esta API devuelve información sobre los mensajes publicados en la red social que cumplan los parámetros de búsqueda introducidos. Lo que se busca es recoger los tweets más recientes de las 10 ciudades españolas más pobladas dentro de la Península Ibérica y las Islas Baleares [34]. Twitter permite realizar una búsqueda filtrando por la ubicación geográfica del usuario en el momento de escribirlo en caso de que este la activara previamente, así como el idioma del mensaje. Por tanto, se necesitan las coordenadas geográficas de las ciudades escogidas. Gracias a la web **latlong.net** se recogen los datos y se genera un archivo JSON que incluye un array de objetos como el mostrado a continuación:

```
{
    name: "Madrid",
    lat: 40.416775,
    lon: -3.703790,
    lang: "es"
}
```

Figura 4-1 Ejemplo de información de ciudad en archivo 'cities.json'.

Esta información se guarda en un archivo llamado "cities.json" que se incluye en la carpeta del código.

Para poder realizar la consulta a la API de Twitter es necesario estar autenticado con unas credenciales generadas por Twitter a partir de nuestro usuario [35]. Al generarlas, Twitter mostrará dos cadenas de texto, "consumer key" y "consumer secret". Estas cadenas deben ser tratadas como contraseñas, ya que a partir de ellas se generará una cadena que se enviará a Twitter a través de una petición HTTP que se responderá, de permitirse el acceso, con un código de acceso. Este código de acceso irá incluido en todas las peticiones que se realicen a Twitter para solicitar los datos.

Este sistema es similar en las API de otras redes sociales, como Instagram [36] o Facebook [37] si bien puede requerirse una autorización previa por parte del sitio web, que deberá revisar el cumplimiento de la aplicación con las condiciones de uso.

Una vez obtenido el código de acceso, se realiza una consulta con los siguientes parámetros, especificados en la documentación de la API de Twitter [38], tomando los datos del archivo "cities.json" (mostrados en mayúscula entre comillas angulares):

```
{
    geocode: <LAT>, <LON>,25km,
    lang: <LANG>,
    count: 100
}
```

Figura 4-2 Ejemplo de consulta realizada a la *API* de Twitter.

Esta petición devolverá un objeto cuyos atributos están descritos en la referencia, de los que sólo interesará almacenar el atributo "text", que contiene el texto del mensaje publicado. Debido a los parámetros aplicados en la consulta, se recibirán 100 mensajes (máximo permitido por consulta) escritos en el idioma <LANG> y geolocalizados hasta 25 kilómetros desde la coordenada marcada. Se genera un objeto con los datos de la ciudad y el texto concatenado de los mensajes y se envía al Agente de carga de datos en bruto. Acto seguido, el script esperará un tiempo predefinido para volver a realizar el proceso. Debido a que la API de Twitter está limitada a un número de consultas por ventana de tiempo, se debe tener en cuenta el número de consultas a realizar y el tiempo de espera para no sobrepasar este número.

La imagen Docker que contendrá esta aplicación parte del archivo Dockerfile siguiente:

```
FROM node:8.11-alpine     # Extiende la imagen node, versión 8.11, variante Alpine
LABEL author="Francisco A. Vargas"     # Datos del autor de la imagen

WORKDIR /app                      # Fija /app como directorio principal
ADD src/package.json /app         # Añade el archivo package.json al contenedor
RUN npm install                   # Descarga las dependencias necesarias
ADD src/ /app                     # Añade el resto de los archivos al contenedor

CMD ["node", "app.js"]            # Ejecuta el script de node.js en /app/app.js
```

Figura 4-3 Instrucciones en archivo Dockerfile para generación de imagen.

y en el archivo de configuración Docker-compose.yml se especifican:

- Las cadenas de autenticación "consumer key" y "consumer secret"
- El nombre del archivo que contiene información de las ciudades ('cities.json')
- El tiempo de espera en milisegundos,
- El nombre que define al agente en el sistema (en este caso 'twitter-crawler')

4.1.2 Agente de carga de datos en bruto

Este agente levanta un servidor que escucha las peticiones del agente de recogida de datos, que llevan la información de la ciudad y el texto en bruto, y tras añadir metadatos, los carga en la base de datos. Por tanto, este módulo necesita conexión con la base de datos y escuchar peticiones HTTP. Para ello, se ha usado Mongoose y Express respectivamente.

Mongoose [39] maneja la comunicación con un servidor MongoDB, y una vez conectado, permite la transmisión de documentos entre la aplicación y la base de datos.
Se implementa un Schema que especifica los tipos de atributos que el objeto que se transmitirá al servidor. Si bien la filosofía de MongoDB y las bases de datos NoSQL no restringen los campos que contiene cada objeto en una colección, Mongoose usa la especificación del objeto definido en el Schema para realizar validación de los datos, además de dar al desarrollador una referencia de qué atributos y de qué tipo deben ir en el objeto a transmitir.

El Schema usado define el objeto con los siguientes parámetros:

```
{
	text: El texto concatenado de los tweets
	date: La fecha actual de generación de este objeto
	city {
		name: Nombre de la ciudad
		lang: Idioma de los mensajes
		loc: Coordenadas de la ciudad
	}
	crawler: Agente de recogida de datos que ha realizado la petición
	status: 0 si el dato aún no ha sido analizado, 1 si lo ha sido
}
```

Figura 4-4 Schema definido para el tipo de dato en bruto.

En cada petición al servidor realizada por el agente de recogida, se creará un objeto del tipo especificado anteriormente y se cargará en la base de datos. Si la carga ha sido satisfactoria, se responderá al agente de recogida con un mensaje de éxito. Por el contrario, si no ha podido realizarse, se responderá con un mensaje de error.

Para la creación de la imagen Docker de este módulo se ha utilizado un archivo Dockerfile similar, al usado en el Agente de recogida de datos, si bien este incluye la instrucción EXPOSE:

```
ADD src/ /app      # Añade el resto de los archivos al contenedor

EXPOSE 3000  # Permite comunicación a través del puerto 3000 entre contenedores.

CMD ["node", "app.js"]    # Ejecuta el script de node.js en /app/app.js
```

Figura 4-5 Extracto del Dockerfile del Agente de carga de datos en bruto.

De esta forma, los contenedores de la misma red virtual creada por Docker podrán enviar peticiones *HTTP* al contenedor a través del puerto 3000 y este las recibirá. En el script de Node.js se inicia un servidor en este puerto.

En el archivo de configuración docker-compose.yml ha sido necesario añadir como variables de entorno el usuario y contraseña del servidor de base de datos.

4.1.3 Servidor de base de datos

El servidor de base de datos utilizado es MongoDB. Dado que no es necesario realizar modificaciones a la imagen publicada por los desarrolladores de MongoDB, ha bastado con importar la imagen y crear un contenedor basado en ella. Sin embargo, para añadir seguridad a su funcionamiento, se ha creado un usuario y contraseña de administrador, que debe ser añadida a todos los módulos que se comuniquen con la base de datos. Según las especificaciones de la imagen MongoDB [40] las variables de entorno

- MONGO_INITDB_ROOT_USERNAME
- MONGO_INITDB_ROOT_PASSWORD

Deben ser definidas en el archivo docker-compose.yml para crear un usuario y asignarle una contraseña, respectivamente.

4.1.4 Agente de gestión de base de datos para el analizador

Este módulo comunica la base de datos con el analizador, por lo que necesita la librería Mongoose [39] para comunicarse con MongoDB, del mismo modo que el Agente de carga definido anteriormente. Utiliza el Schema usado para los objetos en bruto, además de implementar uno para los objetos procesados que recibe del analizador.

Una vez conectado con la base de datos, se ejecuta una función que busca en la colección de datos en bruto de la base de datos si hay objetos del agente de recogida escogido (en este caso 'twitter-crawler') que aún no hayan sido analizados (es decir, cuyo 'status' sea 0).
Una vez recibidos, envía los textos al analizador como petición HTTP, que es respondida con un objeto JSON como se ve en este ejemplo:

```
{
    anger: 4,
    fear: 4,
    joy: 12,
    sadness: 7
    total: 943
}
```

Figura 4-6 Ejemplo de respuesta del analizador de sentimientos.

El agente de gestión añadirá esta información a la recibida por la base de datos, y creará un objeto con esta en una colección diferente, 'processed'. Al finalizar, actualizará el campo 'status' para que el texto no vuelva a ser analizado.

El agente realiza el proceso cada minuto.

Para generar la imagen de este módulo se ha usado un Dockerfile idéntico al usado en el Agente de recogida de datos en bruto, ya que este agente no implementa ningún servidor.

Por otro lado, en el archivo docker-compose.yml se añaden las credenciales de usuario del servidor de base de datos como variables de entorno para este contenedor.

4.1.5 Analizador de sentimientos en texto

El analizador [2], si bien ya está desarrollado, es necesario adaptarlo a su uso en una arquitectura cliente-servidor que le permita recibir peticiones del agente de gestión de base de datos y responda con los resultados.
Para ello, dado que el analizador está desarrollado en Python, se ha usado la librería Flask [24], que permite escuchar peticiones HTTP, similar a la librería Express para node.js usada en varios de los módulos de la plataforma. Cuando el servidor recibe una petición HTTP con un bloque de texto, se ejecuta la función que analiza el texto, y se responde la petición con el resultado de dicha función, como muestra la figura 4-6.

Al tratarse de un contenedor que ejecuta un script de Python, se ha usado la imagen oficial de Python [41], en su versión 2.7 por motivos de compatibilidad con el analizador, y en su variante Alpine, para reducir el tamaño final de la imagen. Igual que se ejecuta el comando 'npm install' para descargar e instalar las dependencias en una aplicación en node.js, se utiliza el comando 'pip install' para instalar las dos dependencias de la aplicación: NTLK para el funcionamiento del analizador y Flask para habilitar su uso como servidor. Por tanto, el archivo Dockerfile es el siguiente:

```
FROM python:2-alpine            # Extiende la imagen python, versión 2-alpine
RUN pip install -U nltk flask   # Instala las dependencias NTLK y Flask

ADD src/ /var/analizador        # Añade el código del analizador
WORKDIR /var/analizador         # Fija el directorio como inicial
ENV FLASK_APP servidor.py       # Fija variable de entorno para iniciar Flask

EXPOSE 5000                     # Permite conexiones entrantes intercontenedor

CMD flask run --host=0.0.0.0 --port=5000      # Inicia el servidor
```

Figura 4-7 Archivo Dockerfile de la imagen del módulo analizador de sentimientos

4.1.6 Servidor API

Para que el cliente pueda acceder a los datos procesados, se implementa este módulo, que hace de puente entre la base de datos y la web. Se desarrolla una aplicación en node.js, usando Express como *framework* para crear el servidor [42], y Mongoose para realizar las consultas a la base de datos. El único punto de acceso a la API es /getCiudades, que realizará una consulta a la base de datos para recibir los valores procesados por el analizador, agrupados por ciudad, y filtrados de tal modo que sólo se reciban los de la última hora, últimas 24 horas o últimos 7 días, en función de un parámetro definido en la petición GET (es decir, /getCiudades?time=hour|day|week). MongoDB dispone de una sintaxis para realizar esta consulta y recibir los datos del modo deseado sin necesidad de manipularlos, gracias a la función 'aggregate' y los parámetros '$match', '$group' y '$sum'. [43]

Para que Mongoose pueda realizar la consulta, se incluye en el código el Schema del tipo de dato procesado que ya se incluye en el módulo Agente de gestión de base de datos para el analizador.

El archivo Dockerfile es idéntico al usado en el módulo Agente de carga de datos en bruto, ya que igual que este, debe implementar un servidor en node.js.

En el archivo Docker-compose.yml, se añaden como variables de entorno las credenciales de usuario de la base de datos. Además, en este contenedor se realiza una conexión de puertos entre el host y el contenedor, ya que los usuarios finales deben poder acceder a este servidor desde fuera de la máquina física. Por tanto, se configura para que al acceder a la URL `http://ip-maquina-fisica:3000`, la máquina física redirija esta petición al contenedor. Se debe tener en cuenta que sólo un contenedor o proceso puede ocupar un puerto de la máquina física a la vez, por lo que si al iniciar la red de contenedores ya existe un servidor escuchando en este puerto el proceso de inicio fallará.

4.1.7 Cliente web

El cliente web es el que realizará las peticiones al servidor API y mostrará los datos recibidos en un panel. Para ello, se instala en la máquina de desarrollo el módulo de node.js 'vue-cli', [27] que construye la estructura básica, también llamada *scaffold*, para desarrollar un sitio web basado en Vue.

Vue implementa un DOM virtual [44], permitiendo un rendimiento superior y una mayor funcionalidad a la hora de modificar la estructura de la página web. Además, permite un enlace de datos bidireccional entre la vista y la lógica, por lo que no será necesario implementar eventos que detecten el cambio en los datos, como en jQuery [45] . Esto resulta en un código más limpio y claro.

Por último, Vue permite dividir en componentes los diferentes bloques con los que se compone la web, creando una jerarquía donde el componente padre puede pasar datos a los componentes hijo, y estos emitir eventos que escuchará el padre. Aprovechando el enlace de datos automático, si los datos cambian en el componente padre, y este tiene un componente hijo con los datos conectados, automáticamente el cambio se registrará en el componente hijo. Por otro lado, cada componente tiene su parte de estructura HTML, el código JavaScript con su estructura de datos local y una hoja de estilos CSS que sólo afecta al propio componente, haciendo más modular el desarrollo de la web.

El panel, usando la librería Axios [46] realiza una petición al servidor API con los datos actualizados, y una vez recibidos los envía a tres componentes, que implementan un tipo de visualización de datos diferente, mostrándose al mismo tiempo. Son los siguientes:

Geográfico:

Utilizando las coordenadas de las ciudades descritas en el archivo 'cities.json' del módulo Agente de recogida de datos en bruto, incluidas en los datos procesados por el analizador, este componente procesa los datos convirtiéndolos en objetos de tipo GeoJSON [47], que se componen de unas coordenadas y propiedades adjuntas al objeto. De este modo, el nombre de la ciudad y los resultados obtenidos en el analizador se añaden al punto como propiedades, y a partir de las coordenadas se sitúa en el mapa utilizando la librería de visualización de

datos D3, que permite manipular datos para construir elementos SVG que se pueden incluir en una web y poder interactuar con ellos [48]. Para poder ubicar el punto en el mapa de una forma más cómoda, se utiliza un objeto GeoJSON con la figura de España peninsular y las Islas Baleares, quedando de esta forma claro dónde está situada cada ciudad. Este gráfico se basa en el tutorial creado por Andy Woodruff para Maptime Boston [49]. El punto cambiará de color en función de la emoción más obtenida por el analizador para esa ubicación, tomando en cuenta la siguiente guía:

Alegría	Verde
Tristeza	Gris
Enfado	Rojo
Miedo	Azul

Tabla 3: Relación entre colores en el mapa y la emoción representada.

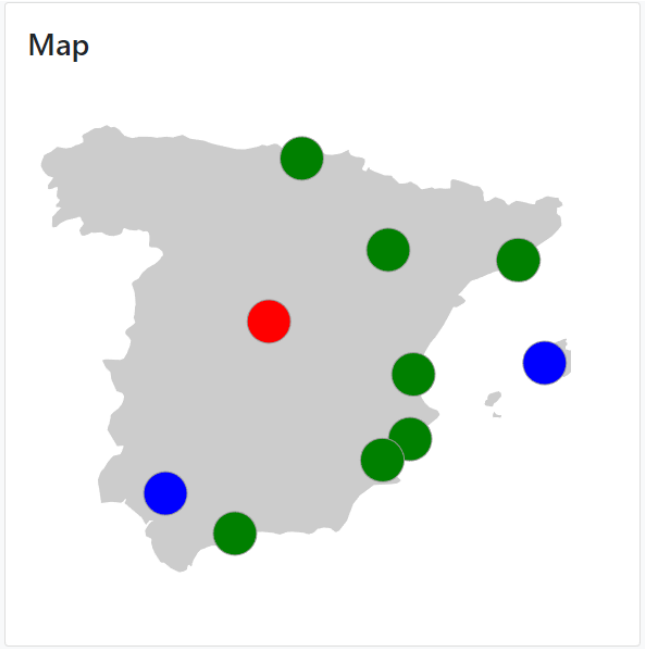

Figura 4-8 Mapa mostrando los resultados del analizador de sentimientos [2].

Gráfico de radar:

Usando los mismos datos que el componente padre solicitó al servidor API y que el subcomponente de Mapa ha transformado para generar un mapa geográfico, la librería chart.js [50] los utiliza para generar un gráfico de radar. Este tipo de gráfico crea un polígono de N vértices, uno por cada atributo a medir, y dibuja un polígono nuevo contenido en el primero, donde los vértices se acercarán más o menos al vértice exterior en función del tamaño del atributo definido en ese vértice. Al realizar una prueba con todas las ciudades, el gráfico se mostraba demasiado abultado, por lo que se decidió cambiar la visualización. Interceptando el evento 'onMouseOver' de JavaScript, la librería D3.js puede detectar en el mapa si el ratón del usuario está sobre alguna ciudad analizada, enviando esta información a través de un evento al componente padre, que lo transmitirá al componente hijo que dibuja el gráfico de radar. De esta forma, el gráfico de radar dibujará el polígono de la ciudad

seleccionada únicamente, para desaparecer en caso de que el usuario deje de situarse encima de la ciudad.

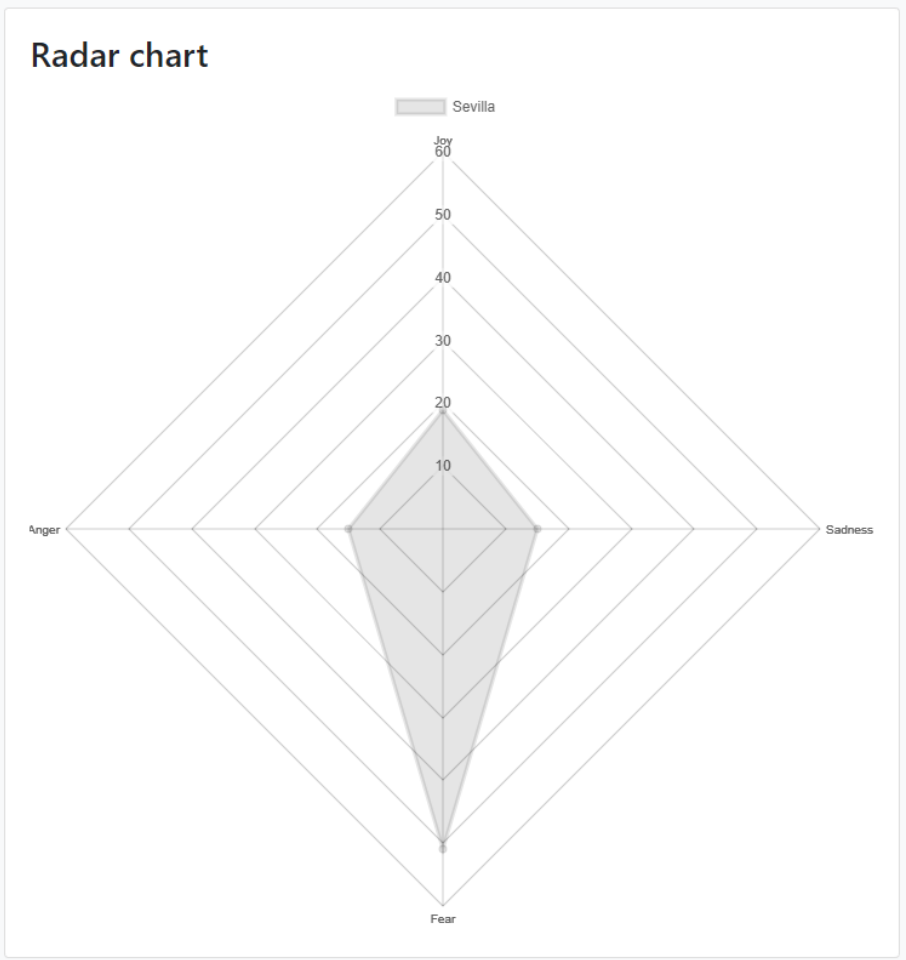

Figura 4-9 Detalle del panel Gráfico de radar del cliente web.

Tabla:

Para poder mostrar los datos analizados de una forma numérica, se ha usado una tabla tradicional en HTML. Sin embargo, esta tabla reacciona de la misma forma que el gráfico de radar a la ciudad sobre la que el usuario está pasando el ratón, mostrando un símbolo en la fila de la ciudad seleccionada. De esta forma, incluso aquellos usuarios que no conozcan las ciudades y su ubicación geográfica podrán conocer su nombre y sus datos. La tabla mostrará el número total de palabras analizadas por ciudad y el porcentaje de estas que reflejan alegría, tristeza, miedo y enfado. El color de fondo de cada fila viene definido por la emoción predominante en la ciudad. Si se pasa el ratón encima de un porcentaje aparece el dato absoluto de palabras encima de su porcentaje.

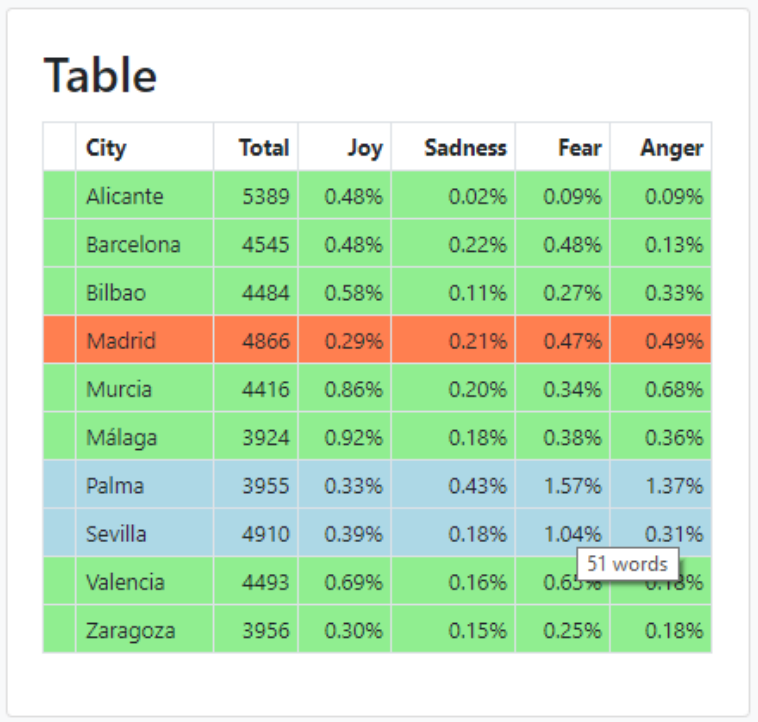

Figura 4-10 Detalle del panel Tabla del cliente web.

Además de la modularidad que ofrece Vue con sus componentes, se ha usado el componente 'vue-router' [51] para desarrollar una *SPA* o 'Single Page Application' de una forma sencilla. Para entender este concepto, se recuerda que, en un sitio web tradicional, cada vez que se pulsa en un enlace, el servidor genera una página completa y la envía al cliente, que la renderizará desde cero, aunque la mayoría de sus elementos no hayan cambiado (como barras laterales o superiores, o archivos de imagen, estilos o scripts). Con 'vue-router', todos los componentes del sitio web son cargados en la primera visita a la página, pero al pulsar un enlace, no será el servidor quien genere la página nueva, sino el motor JavaScript, que eliminará de la pantalla los componentes que ya no deban mostrarse, y cargará aquellos que sí. Por supuesto, los datos no están cargados en esta página, pero una vez cargada los solicitará al servidor a través de la *API*, que sólo enviará los datos (en formato *JSON*) y no el código HTML completo. Este patrón de diseño se apoya en el cliente para realizar el trabajo de cambiar de página, dejando al servidor sólo la labor de servir los datos que la alimentan. Sin embargo, los dispositivos que usa el cliente para navegar son lo suficientemente potentes como para que esto suponga un problema.

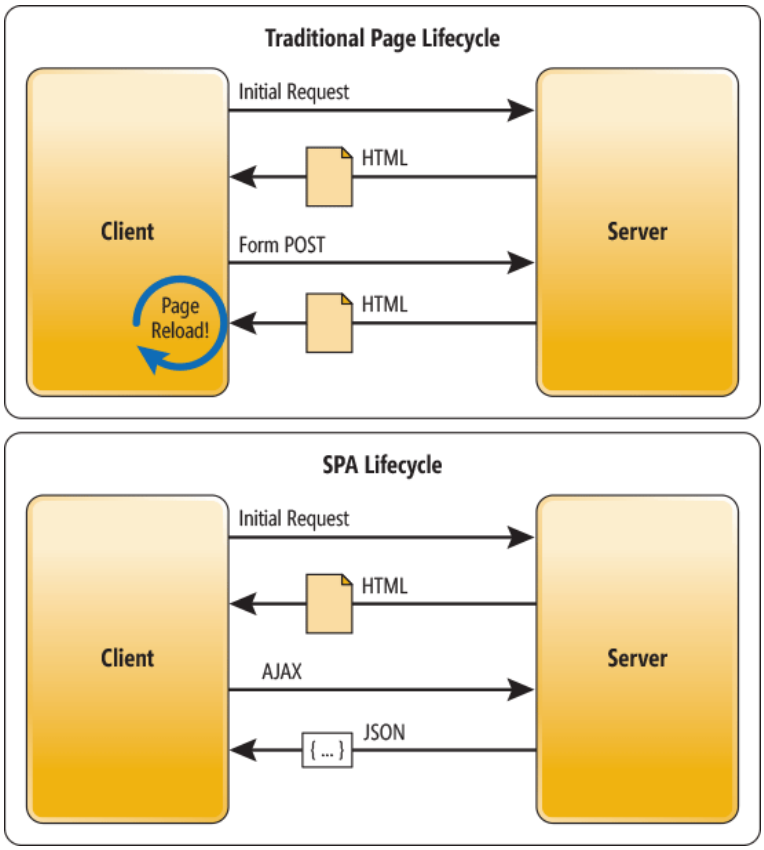

Figura 4-11 Diagrama de ciclo de vida de web tradicional vs *SPA*. [52]

La desventaja en este tipo de páginas es que los *bots* o programas que usan los buscadores para indexar el contenido de la web no están preparados para este tipo de páginas [53] por lo que se deben utilizar módulos que realicen el renderizado de la página en el servidor, por lo que, aunque Google está mejorando su capacidad de indexado en SPA, hay que realizar un control más fino de cómo ven la aplicación web los *bots* de los buscadores.

Un detalle a tener en cuenta en el desarrollo de webs con *frameworks* basados en JavaScript es que se pueden llegar a usar muchos módulos de terceros para montar la aplicación. Tradicionalmente esto suponía incluir cada módulo en el apartado <head> de la página HTML, y alojando cada uno de ellos en el servidor o dirigiendo al navegador a un CDN o Content Delivery Network para que lo descargara. Con las hojas de estilo CSS ocurría algo parecido. Sin embargo, la herramienta Webpack, dependiente de node.js, [54] permite empaquetar cada módulo JavaScript utilizado en la web y juntarlo en un solo archivo, que además puede ser optimizado para reducir su tamaño y tiempo de descarga. De esta forma, es posible consolidar todos los scripts en un solo archivo JavaScript, que será el único a incluir en el servidor y en el código HTML, y el único que tendrá que descargar el usuario.

En la fase de desarrollo, es posible activar Webpack de tal modo que se ejecute cada vez que un archivo cambie, realizando la operación de empaquetado cada vez. Webpack permite el uso de plug-ins para procesar y transpilar, es decir, convertir en otro lenguaje, diferentes fuentes de código, por ejemplo, de SASS a CSS, o de TypeScript a JavaScript.

Una vez preparado para producción, se ejecuta Webpack en modo producción, que generará los archivos finales a alojar en un servidor. Como estos archivos sólo incluyen JavaScript

que se ejecutará en el cliente, imágenes y código HTML y CSS, es posible alojar estos archivos en un servidor de archivos estáticos.

Por tanto, la imagen que se genera para este módulo no está basada en Node.js, sino en Nginx [55], un servidor web, al que se le incluyen los archivos previamente generados en la imagen. Finalmente, se enlaza el contenedor ejecutado con la máquina host a través del puerto 80, de tal forma que un usuario que acceda a `http://ip-maquina-fisica` acceda a este contenedor.

Nótese que en el código del cliente web se realizarán peticiones al servidor API, pero estas peticiones se realizarán desde el ordenador del usuario, por lo que la dirección a la que deben ir dirigidas es `http://ip-maquina-fisica:3000`, ya que en este caso no se trata de comunicación intercontenedor, en la que se pondría el nombre del contenedor en la URL.

4.1.8 Agente de copia de seguridad de base de datos

Por último, se ha desarrollado un módulo que realiza una copia de seguridad de la base de datos. Para ello, se ha creado una imagen basada en Alpine a la que se le han añadido los módulos curl, mongodb-tools y xz. El primero permite realizar peticiones HTTP, con la que se podrá enviar un mensaje al panel de control de estados. El segundo provee la herramienta 'mongodump', aplicación oficial de MongoDB para realizar copias de seguridad de sus bases de datos. El tercero es un compresor que desde el año 2014 ha reemplazado a bzip2 por la organización del Kernel Linux [57] para empaquetar los archivos. xz provee de mejor compresión, aunque usa más memoria para realizar la descompresión que otras herramientas, como gzip o bzip2 [58]. Sin embargo, esto no es un problema para los archivos que se van a comprimir, así como el hardware donde se va a ejecutar.

Se ha escrito un script en SH que realiza la siguiente tarea:

- Fija en una variable la fecha y hora actual en formato YYYYMMDD_HHMMSS para nombrar a los archivos que serán generados en este proceso
- Ejecuta 'mongodump' conectándose al servidor de base de datos, usando las credenciales de acceso fijadas en el archivo docker-compose.yml y especificando un directorio de salida auxiliar
- Una vez realizado el volcado de datos, se empaqueta la carpeta generada en un archivo tar.
- A continuación, se realiza la compresión XZ del empaquetado, permitiendo al proceso utilizar todos los núcleos del servidor, ya que permite paralelización.
- Se mueve el archivo XZ generado a la carpeta que ha sido especificada en el archivo Docker-compose.yml dentro del apartado 'volumes', de tal forma que el archivo quedará fuera del sistema de archivos del contenedor.
- Se borran los directorios y archivos temporales y se muestra un mensaje de éxito, que será recogido por Fluentd
- Se envía una petición HTTP al servidor de control de estado usando curl.
- Se suspende el proceso durante un día.

4.1.9 Panel de control de estado

Se ha creado un sencillo panel de control, con el objetivo de poder controlar el funcionamiento de cada módulo desde una página web, sin necesidad de acceder a la máquina host a través de SSH.

Cada módulo (salvo la base de datos y el cliente web) envía una señal al servidor de panel de control cada vez que realizan la acción para la que fueron creados y ejecutados. El servidor, cuando se le solicita esta información, muestra una web donde cada módulo es representado en un recuadro, donde figuran su nombre y el tiempo de la última conexión. Dado que los tiempos de espera están predefinidos en el archivo docker-compose.yml, el administrador puede detectar si algún módulo está fallando, y acudir a los logs generados por el contenedor y recogidos por Fluentd para comprobar qué está ocurriendo.

Para el desarrollo de este módulo se ha elegido el framework PHP Laravel [26], que permite un desarrollo muy sencillo de aplicaciones web, además de tener integrado el uso de Webpack y Vue para el desarrollo del front-end.

Para el almacenamiento de datos se usa una base de datos SQLite, dado que el uso de un servidor MySQL o PostgreSQL excede las necesidades de esta aplicación, que almacena una fila por módulo y dos campos por fila (nombre y última fecha). SQLite no es un servidor de base de datos, sino que a través de una librería realiza las operaciones de base de datos sobre un archivo. [56].

Una vez desarrollado, Front-end y Back-end se alojan en el mismo servidor. Se ejecuta Laravel-mix (la versión de Webpack creada por el equipo de Laravel) para generar los archivos JavaScript y CSS finales. Del mismo modo que NPM y Pip en Python, PHP usa Composer para descargar todas las dependencias necesarias para el funcionamiento de Laravel, que en este caso se hará en el equipo de desarrollo en vez de dentro de la imagen.

Una vez se tenga el directorio completo, se procede a crear la imagen, usando la imagen base de Ubuntu a la que se le añaden los paquetes necesarios para poder ejecutar código PHP. Finalmente, se añade el directorio con la aplicación web y se manda a ejecutar el servidor web integrado en PHP cuando el contenedor sea creado.

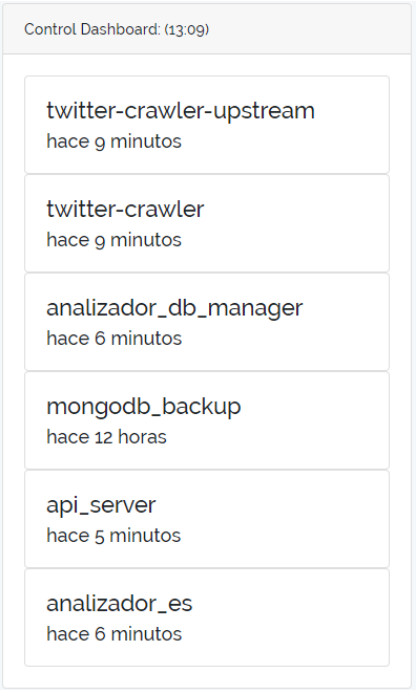

Figura 4-12 Captura de pantalla del Panel de control de estado.

5 Integración, pruebas y resultados

5.1 Integración

Una vez se han desarrollado todas las imágenes, es necesario describir un archivo de configuración docker-compose.yml que desplegará la arquitectura. [59]
Se tiene especial atención en:

- El nombre que tendrá cada contenedor, ya que aquellos que deben comunicarse lo harán a través de este nombre, que Docker convertirá en la IP de la red virtual creada para comunicación intercontenedor.
- Las variables de entorno que cada contenedor necesita para funcionar deben estar definidas correctamente
- Aquellos contenedores que requieren entrada del exterior (en este caso, el servidor API, el cliente web, el panel de control de estados y el servidor Fluentd), que deben tener un puerto asociado al host y otro en el contenedor para que Docker pueda realizar el mapeo. Estos puertos deben ser únicos en el lado de host, y estar libres a la hora del despliegue.
- Aquellos contenedores que requieren el uso de volúmenes deben tener especificado el par <directorio en host>:<directorio en contenedor>

Se puede establecer una jerarquía entre los contenedores, de tal forma que un contenedor esperará a uno o varios contenedores especificados en el atributo "depends_on" para iniciarse. Esto es muy útil si un contenedor va a conectarse a un servidor, y este servidor también se iniciará en el mismo momento. Sin embargo, esta restricción no funciona perfectamente en algunos servidores, ya que Docker no es capaz de detectar cuando ha dejado de iniciarse y está dispuesto a recibir peticiones.

Con el archivo de configuración finalizado, se ejecutaría el comando 'docker-compose up' para desplegar la plataforma, con la terminal ubicada en el directorio principal del proyecto. Este comando crea una red virtual que conecta a todos los contenedores descritos entre sí y con el host, construye las imágenes a partir de sus archivos Dockerfile si se especifica, o la descarga del repositorio Docker Store, a no ser que ya se hayan generado o descargado previamente, y creará e iniciará un contenedor para cada servicio descrito, mostrando los registros que cada uno vaya generando.

Sin embargo, dado que se usa Fluentd para manejar los registros, Docker advertirá que no puede mostrar los mensajes por pantalla, por lo que se usa el comando 'docker-compose up -d' para que este se ejecute en segundo plano.

Como se ha advertido en la lista anterior, es posible que la lista de dependencias no sea suficiente a la hora de iniciar la plataforma de forma escalonada, iniciando primero aquellos servidores de los que dependen otros contenedores. Por tanto, se realiza el despliegue en varias fases:

- Fase 1: Se despliega Fluentd, ya que es una dependencia de todos los contenedores para manejar sus registros:

```
docker-compose -d up fluentd
```

- Fase 2: Se despliega la web de panel de control, el servidor de base de datos MongoDB y el analizador de datos:

  docker-compose -d up logger_server mongodb analizador_es

- Fase 3: Una vez iniciada la base de datos, se despliegan los contenedores que dependen de ella:

docker-compose -d up analizador_db_manager crawler_upstream mongodb_backup api_server

- Fase 4: Por último, se despliegan los contenedores de recogida de datos y el cliente web:

  docker-compose -d up crawler web_client

Ejecutando el comando docker-compose ps se muestra una lista de los procesos que está ejecutando cada contenedor.

Si se trata del primer despliegue, en unos minutos aparecerán los primeros datos en la web, así como en el panel de control. Si se trata, sin embargo, de una parada por mantenimiento, habrá que cambiar el comando up por start, para el que no hará falta usar el parámetro -d.

5.1.1 Modificaciones para adaptar la plataforma a otras fuentes o analizadores

A continuación, se explica cómo se debe actuar para modificar la plataforma, en función de la modificación requerida:

¿Qué hacer sí…

…se desea mantener la plataforma, pero modificando las ciudades?	Modificar el archivo 'cities.json' ubicado en el Agente de recogida de datos en bruto, añadiendo las ciudades deseadas.
…se desea mantener la plataforma, pero realizando otras consultas para mostrar en el cliente web?	Añadir diferentes puntos de entrada al servidor API, donde se realizará la consulta a la base de datos. Crear un componente en el cliente web que solicite estos datos y los muestre por pantalla.
…se desea mantener la plataforma, pero cambiando el analizador?	Si el analizador está ubicado en un servicio web existente, el Agente de gestión de datos al analizador debe realizar la petición y manejar la respuesta, manipulando los datos que se enviarán al analizador para que sea compatible con el servicio, así como la respuesta, para que sea compatible con el servidor API. Si el analizador es un software propio, debe adaptarse a la interfaz cliente-servidor si no lo está, crear una imagen con el módulo adaptado e iniciar un contenedor en la misma red virtual para que se pueda conectar a través de comunicación intercontenedor.
…se desea ejecutar el analizador u otro servidor en otra máquina?	Se creará otro archivo de configuración docker-compose.yml en la nueva máquina con el servicio o servicios que se desea mover, permitiendo su salida al

	exterior a través de puertos en la máquina host, y se fija su dirección IP y puerto como variables de entorno en los contenedores que lo utilicen. Si se desea seguir usando fluentd, es posible añadir una dirección IP del servidor fluentd en el archivo Docker-compose.yml (no necesaria en la plataforma inicial pues está en la misma máquina host) para que Fluentd pueda recoger sus registros
…se desea mantener la plataforma, pero cambiando el Agente de recogida de datos en bruto?	Si el Agente de recogida de datos en bruto recoge textos agrupados por ciudades, simplemente deberá adaptar su salida a la esperada por Agente de carga de datos en bruto, que cambiará el nombre del Agente de recogida para su etiquetado en la base de datos. Posteriormente, habría que añadir el nombre del nuevo Agente de recogida a los servidores que recogen datos de la base de datos para que estos envíen sus datos a los analizadores y clientes web. Si el nuevo Agente de recogida no recoge información de texto agrupada por ciudades, la modificación debe ser más exhaustiva, si bien la base de datos y su agente de copia de seguridad, el panel de control de estado y la estructura del resto de módulos no debe modificarse para su funcionamiento.

Tabla 4 Soluciones propuestas a diferentes formas de modificar la plataforma.

5.2 Pruebas y calidad del software

5.2.1 ESLint

Para mejorar la calidad del código, aprovechando que la mayoría de la plataforma está escrita en JavaScript, se ha utilizado la herramienta ESLint [60] que analiza el código y advierte de posibles errores en el mismo. Además, permite añadir reglas de estilo, que fuerzan al desarrollador a escribir el código en base a un estilo predeterminado. Muchos editores de texto permiten integrar estos mensajes de error y advertencias en la ventana de edición, marcando el error directamente sobre el código JavaScript.

Para este código, se ha utilizado la regla de estilo de standard.js [61] que proporciona unas reglas sencillas por defecto. Hay que destacar que estas reglas de estilo no afectarán al rendimiento del código ni eliminarán fallos, pero unificará la forma de escribir el código para todos los archivos del proyecto, estén escritos por uno o varios desarrolladores. Esto hace más legible la revisión del código.

Como ejemplo, varias reglas que usa standard.js son:

- Usar dos espacios como sangría
- Utilizar comillas simples para definir cadenas de texto
- No declarar variables que no se usan en el código (o, al menos, no se especifica su conexión con otros módulos donde sí puedan ser usadas)
- Añadir un espacio entre la palabra clave ‘if’ y el paréntesis de apertura del condicional.

Se considera el uso de esta herramienta como muy beneficioso para el desarrollo, sobre todo si este se realiza dentro de un equipo de desarrolladores. [62]

5.2.2 Mocky y peticiones HTTP simuladas

Si se desea comprobar el funcionamiento de un cliente antes de tener desarrollado el servidor, es posible utilizar un servicio de peticiones HTTP simuladas. Se puede configurar un servidor propio en el sistema, que se configura para devolver siempre la misma respuesta (eliminando toda la lógica), o se puede hacer uso de un servicio online, como Mocky [63], que permite la creación de respuestas HTTP especificando el contenido, el código de respuesta y el tipo de contenido. Una vez creada, Mocky generará una URL, que podrá ser utilizada en el cliente para realizar la petición. Si el cliente está bien diseñado, la respuesta será correctamente procesada, pudiendo realizar las pruebas antes de tener el módulo real. Mocky podrá incluso responder la petición con un retraso especificado en la URL, para comprobar el funcionamiento del cliente incluso en condiciones de baja calidad de red.

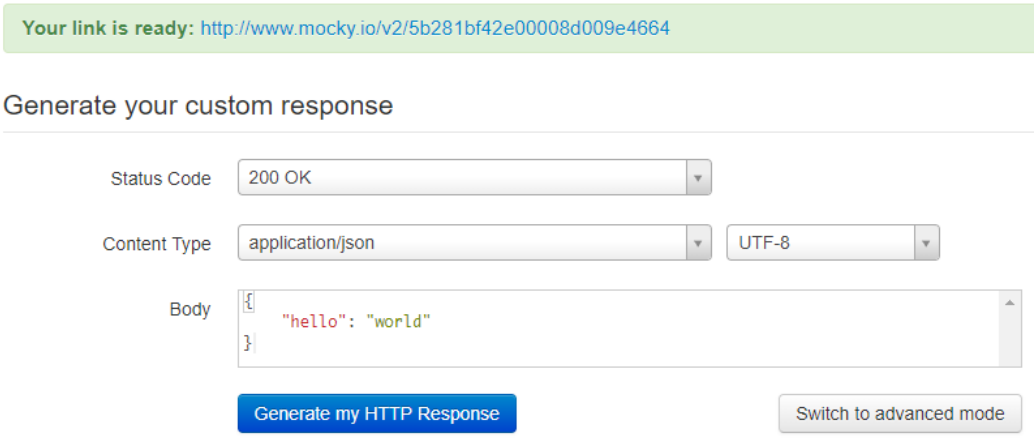

Figura 5-1 Captura de pantalla de Mocky.

Por otro lado, para probar el servidor, una herramienta muy utilizada es Postman. [64] Inicialmente desarrollada como una extensión de Google Chrome, ahora es una aplicación independiente que permite realizar peticiones HTTP a URLs, especificando el tipo de petición (GET, POST, PUT…), las cabeceras que irán con la petición, así como parámetros GET y cuerpo de la petición. Postman realizará la petición y devolverá la respuesta en un panel inferior, formateándola como JSON o HTML si ese fuera el contenido recibido.

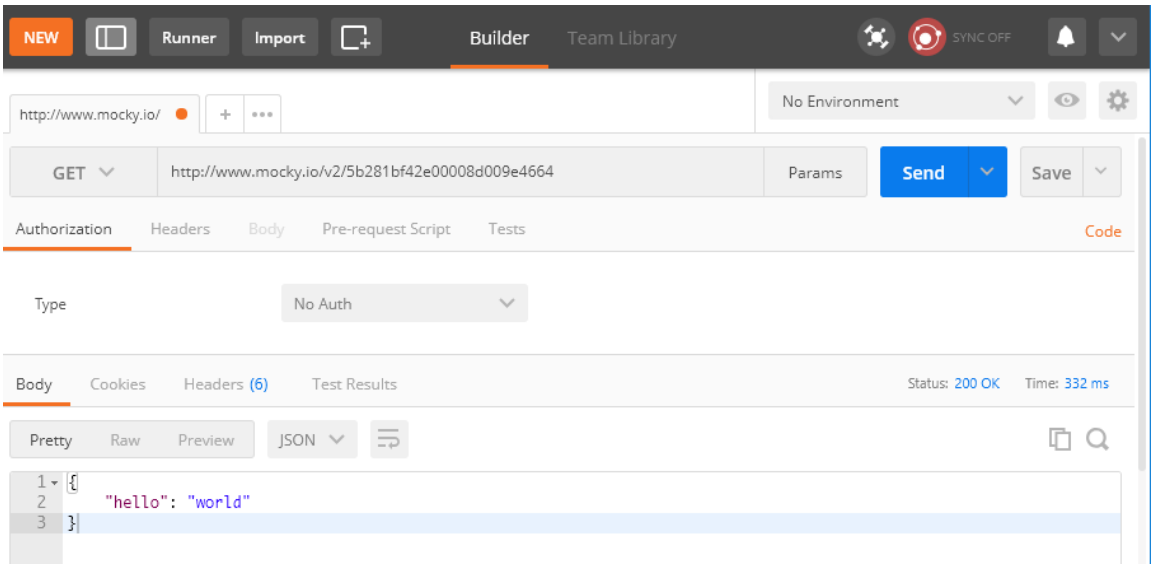

Figura 5-2 Captura de pantalla de Postman.

5.2.3 Panel de desarrollador del navegador y Vue Devtools

Una vez desarrollado el cliente web, el uso del panel de desarrollador del navegador es casi obligatorio, ya que proporciona al desarrollador información detallada sobre las transferencias de datos con el servidor, fallos en la ejecución del código JavaScript e incluso emulación de dispositivos, si se desea desarrollar una web compatible con dispositivos móviles y/o táctiles.

Además de este panel, se puede instalar una extensión al navegador [65] que, mientras se desarrolla la interfaz web con Vue, mostrará datos de esta, con información de los componentes, los datos que contiene cada uno y los eventos que se emiten. Esta extensión sólo funciona con interfaces Vue en desarrollo, no aquellas que han sido generadas en modo producción.

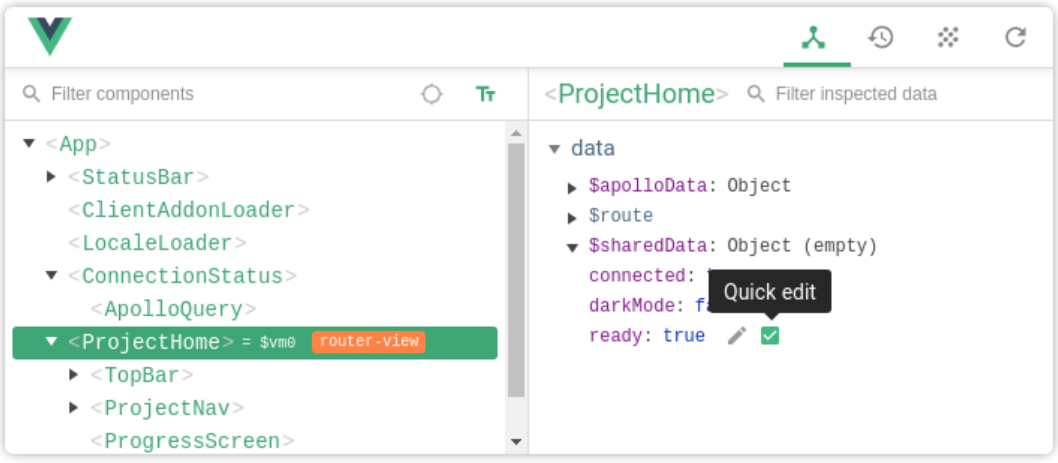

Figura 5-3 Captura de pantalla de Vue Devtools.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Este trabajo ha recogido el conocimiento suficiente para saber cómo realizar una herramienta de visualización de datos, así como saber reconocer en qué situaciones usar una u otra herramienta de desarrollo, basándose en aspectos técnicos y de recursos.

La plataforma cumple con todos los puntos tratados en el apartado de diseño, así como con los requisitos de portabilidad y modularidad.

El uso de contenerización ha permitido mejorar en eficiencia tanto en el desarrollo como en la puesta en producción. Como ejemplo, se ha probado esta plataforma en un VPS de bajo coste, con un sistema operativo GNU/Linux y Docker instalado, y la puesta en producción se realizó en menos de 10 minutos desde el inicio de sesión en la máquina hasta la puesta en marcha de los contenedores.

El uso de herramientas modernas, bien documentadas y de gran acogida por la comunidad ha permitido encontrar bastante información sobre el uso óptimo de estas, así como la respuesta a dudas que han surgido en la fase de desarrollo.

6.2 Trabajo futuro

Si bien el trabajo futuro más inmediato será comprobar cómo se realizan las adaptaciones a la plataforma para su uso con nuevos módulos, y optimizar los procesos para realizarlo de una forma más intuitiva, se proponen dos para mejorar la seguridad y acceso a la misma:

- Actualmente el agente de copia de seguridad almacena el archivo generado en una carpeta del mismo sistema host donde se ejecuta el servidor. Si bien se ha realizado así para evitar usar recursos adicionales, no es una técnica óptima, ya que un fallo en la máquina host haría perder tanto la base de datos en producción como todas las copias de seguridad. Se podría mejorar este Agente (o crear uno paralelo) que además de realizar la copia de seguridad, enviara esta fuera de la máquina a través de un servicio remoto, para así añadir más seguridad a la plataforma.

- El acceso al servidor API, al cliente web y al panel de control se realiza a través de una URL a la misma máquina, pero a diferentes puertos. Algunas configuraciones de seguridad en empresas o centros educativos podrían bloquear todo tráfico que no fluya por los puertos estándar de web (80 para HTTP y 443 para HTTPS), haciendo imposible el uso de la plataforma si se navega desde una de estas redes. Sin embargo, si se asigna un nombre de dominio y se enlaza con la IP de la máquina física, se podría utilizar un proxy inverso para que realizara las funciones de distribuidor dentro de la máquina. [66] De tal forma que cada contenedor con entrada del exterior tuviera un subdominio, como, por ejemplo:

    - `api.dominio.xyz` para el servidor API,
    - `panel.dominio.xyz` para el panel de control
    - `www.dominio.xyz` para el cliente web,

    siendo todas estas comunicaciones transmitidas a través del puerto 80, ya que será el proxy quien se encargará de dirigir el tráfico dentro de la red de la plataforma.

Referencias

[1] G. Brown, «It Works on My Machine! How Container Technologies Like Docker Can Revolutionize Continuous Integration» 20 Junio 2014. https://www.usenix.org/conference/ures14/technical-sessions/presentation/it-works-my-machine-how-container-technologies [Último acceso: 19 Junio 2018].

[2] P. Rodriguez, A. Ortigosa y R. M. Carro, «Detecting and making use of emotions to enhance student motivation in e-learning» de *International Journal of Continuing Engineering Education and Life Long Learning, 24 (2),* pp. 168-183. doi: 10.1504/IJCEELL.2014.060156 ISSN: 1560-4624

[3] V. Bogdanov, «MEAN vs LAMP: Choosing the Right Stack for Your Startup» Built in Chicago, 7 Febrero 2017. https://www.builtinchicago.org/blog/mean-vs-lamp-choosing-right-stack-your-startup [Último acceso: 19 Junio 2018].

[4] J. K. Waters, «Virtualization Definition and Solutions» CIO, 15 Marzo 2007. https://www.cio.com/article/2439494/virtualization/virtualization-definition-and-solutions.html [Último acceso: 19 Junio 2018].

[5] VMware, Inc., «A Perfomance Comparison of Hypervisors» 2007.

[6] J. Wallen, «How to improve Virtualbox guest performance in five steps» Tech Republic, 1 Febrero 2017. https://www.techrepublic.com/article/how-to-improve-virtualbox-guest-performance-in-five-steps/ [Último acceso: 2018 Junio 19].

[7] OVH, «VPS SSD» 19 Junio 2018. https://www.ovh.es/vps/vps-ssd.xml [Último acceso: 19 Junio 2018].

[8] OVH, «Servidores Dedicados» 19 Junio 2018. https://www.ovh.es/servidores_dedicados/bestvalue/ [Último acceso: 19 Junio 2018].

[9] Digital Ocean, LLC., «Digital Ocean - Pricing» 19 Junio 2018. https://www.digitalocean.com/pricing/ [Último acceso: 19 Junio 2018].

[10] S. Mazaheri, Y. Chen, H. Elham y S. Alan, «Cloud benchmarking in bare-metal, virtualized, and containerized execution environments» 19 Diciembre 2016. https://ieeexplore.ieee.org/document/7790286/ [Último acceso: 19 Junio 2018] doi: 10.1109/CCIS.2016.7790286

[11] R. Osnat, «A Brief History of Containers: From the 1970s to 2017» 21 Marzo 2017. https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016 [Último acceso: 19 Junio 2018].

[12] Docker, inc., «Docker Engine - Frequently Asked Questions» Junio 2018. https://docs.docker.com/engine/faq/ [Último acceso: 19 Junio 2018].

[13] Docker, Inc., «Dockerfile reference» https://docs.docker.com/engine/reference/builder/ [Último acceso: 19 Junio 2018].

[14] Docker, Inc., «Docker Store» 2018. https://store.docker.com [Último acceso: 19 Junio 2008].

[15] Datadog, Inc., «8 surprising facts about Docker adoption» Junio 2018. https://www.datadoghq.com/docker-adoption/ [Último acceso: 19 Junio 2018].

[16] Node.js Foundation, «Node.js - About» Junio 2018. https://nodejs.org/en/about/ [Último acceso: 19 Junio 2018].

[17] A. Delgado, «Top 5 companies using NodeJS in production» 5 Octubre 2016. https://www.linkedin.com/pulse/top-5-companies-using-nodejs-production-anthony-delgado/ [Último acceso: 19 Junio 2018].

[18] Stack Overflow, Inc., «Developer Survey Results - 2018» Junio 2018. https://insights.stackoverflow.com/survey/2018#technology [Último acceso: 19 Junio 2018].

[19] SimilarTech, Ltd., «jQuery Market Share and Web Usage Statistics» Junio 2018. https://www.similartech.com/technologies/jquery [Último acceso: 19 Junio 2018].

[20] S. Saunier, «From jQuery to DOM and ES6» 9 Agosto 2017. https://www.lewagon.com/es/blog/from-jquery-to-dom-and-es6 [Último acceso: 19 Junio 2018].

[21] M. Petrosyan, «Angular5 vs React vs Vue» 7 Febrero 2018. https://itnext.io/angular-5-vs-react-vs-vue-6b976a3f9172 [Último acceso: 19 Junio 2018].

[22] npm, inc., «What is npm?» 21 Marzo 2018. https://docs.npmjs.com/getting-started/what-is-npm [Último acceso: 19 Junio 2018].

[23] The Apache Software Foundation, «Apache Cordova» Junio 2018. https://cordova.apache.org/ [Último acceso: 19 Junio 2018].

[24] Electron Community, «Electron» https://electronjs.org/ [Último acceso: 19 Junio 2018].

[25] Docker, Inc., «Docker Compose file reference» https://docs.docker.com/compose/compose-file/ [Último acceso: 19 Junio 2018].

[26] A. Ronacher, «Flask (A Python Microframework)» 2018. http://flask.pocoo.org/ [Último acceso: 19 Junio 2018].

[27] E. You, «Vue CLI» 2018. https://cli.vuejs.org/ [Último acceso: 19 Junio 2018].

[28] T. Otwell, «Laravel, the PHP Framework for Web Artisans» 2018. https://laravel.com/ [Último acceso: 19 Junio 2018].

[29] Docker, Inc., «Best practices for writing a Dockerfile» https://docs.docker.com/develop/develop-images/dockerfile_best-practices/#decouple-applications [Último acceso: 2018 Junio 2018].

[30] Docker, Inc., «Manage data in Docker» https://docs.docker.com/storage/ [Último acceso: 19 Junio 2018].

[31] Fluentd Project, «Fluentd, Open source data collector» https://www.fluentd.org/ [Último acceso: 19 Junio 2018].

[32] Docker, Inc., «Fluentd Image - Docker Store» https://store.docker.com/community/images/fluent/fluentd [Último acceso: 19 Junio 2018].

[33] Docker, Inc., «Node Image - Docker Store» https://store.docker.com/images/node [Último acceso: 19 Junio 2018].

[34] Alpine Linux Development Team, «Alpine Linux» https://alpinelinux.org/ [Último acceso: 19 Junio 2018].

[35] Twitter, Inc., «Twitter Developers» https://developer.twitter.com/en/docs/basics/getting-started [Último acceso: 19 Junio 2018].

[36] Instituto Nacional de Estadística, «Cifras oficiales de población de los municipios españoles: Revisión del Padrón Municipal» 29 Diciembre 2017.

http://www.ine.es/dyngs/INEbase/es/operacion.htm?c=Estadistica_C&cid=1254736177011&menu=resultados&idp=1254734710990 [Último acceso: 19 Junio 2018].

[37] Twitter, Inc., «Twitter Developers - Authentication» 2018. https://developer.twitter.com/en/docs/basics/authentication/overview [Último acceso: 19 Junio 2018].

[38] Instagram, Inc., «Instagram developer documentation» 2018. https://www.instagram.com/developer/ [Último acceso: 19 Junio 2018].

[39] Facebook, Inc., «API Graph - Facebook for Developers» 2018. https://developers.facebook.com/docs/graph-api [Último acceso: 19 Junio 2018].

[40] Twitter, Inc., «Standard Search API - Twitter Developers» 2018. https://developer.twitter.com/en/docs/tweets/search/api-reference/get-search-tweets [Último acceso: 19 Junio 2018].

[41] LearnBoost, «Mongoose - Elegant mongodb object modelling for node.js» http://mongoosejs.com/ [Último acceso: 19 Junio 2018].

[42] Docker Inc., «MongoDB Image - Docker Store» https://store.docker.com/images/mongo [Último acceso: 19 Junio 2018].

[43] Docker, Inc., «Python Image - Docker Store» https://store.docker.com/images/python [Último acceso: 19 Junio 2018].

[44] node.js Foundation, «Express.js - Node.js web application framework» http://expressjs.com/ [Último acceso: 19 Junio 2018].

[45] MongoDB, Inc., «Aggregation - MongoDB Manual» https://docs.mongodb.com/manual/aggregation/ [Último acceso: 19 Junio 2018].

[46] A. Gore, «What's The Deal With Vue's Virtual DOM?» 21 Febrero 2017. https://vuejsdevelopers.com/2017/02/21/vue-js-virtual-dom/ [Último acceso: 19 Junio 2018].

[47] A. Gore, «Exploring Vue.js: Reactive Two-Way Data Binding» medium.com, 19 Septiembre 2016. https://medium.com/js-dojo/exploring-vue-js-reactive-two-way-data-binding-da533d0c4554 [Último acceso: 2018 Junio 2018].

[48] «Axios - Github» https://github.com/axios/axios [Último acceso: 19 Junio 2018].

[49] «GeoJSON Specification» http://geojson.org/ [Último acceso: 19 Junio 2018].

[50] M. Bostock, «D3.js - Data Driven Documents» https://d3js.org/ [Último acceso: 19 Junio 2018].

[51] A. Woodruff, «Mapping with D3» Maptime Boston, http://maptimeboston.github.io/d3-maptime [Último acceso: 19 Junio 2018].

[52] Comunidad Chart.js, «Chart.js - Open Source HTML5 charts for your website» http://chartjs.org/ [Último acceso: 19 Junio 2018].

[53] «Vue Router» https://router.vuejs.org/ [Último acceso: 19 Junio 2018].

[54] A. Alonso, «Single Page App vs Multi Page App» 2 Febrero 2015. https://adrianalonso.es/desarrollo-web/single-page-app-vs-multi-page-app/ [Último acceso: 19 Junio 2018].

[55] A. Gore, «Is My Single-Page Application SEO Friendly?» 9 Abril 2018. https://vuejsdevelopers.com/2018/04/09/single-page-app-seo/ [Último acceso: 19 Junio 2018].

[56] «Webpack» https://webpack.js.org/ [Último acceso: 19 Junio 2018].

[57] «nginx» http://nginx.org/en [Último acceso: 19 Junio 2018].

[58] Linux Kernel Organization, Inc., «The Linux Kernel Archives» 27 Diciembre 2013. https://www.kernel.org/happy-new-year-and-good-bye-bzip2.html [Último acceso: 19 Junio 2018].

[59] J. Farncomb, «13 simple xz examples» 10 Septiembre 2015. https://www.rootusers.com/13-simple-xz-examples/ [Último acceso: 19 Junio 2018].

[60] «SQLite» https://www.sqlite.org/about.html [Último acceso: 19 Junio 2018].

[61] JS Foundation, «ESLint - Pluggable JavaScript Linter» https://eslint.org/ [Último acceso: 19 Junio 2018].

[62] F. Aboukhadijeh, «JavaScript Standard Style» https://standardjs.com/ [Último acceso: 19 Junio 2018].

[63] F. Aboukhadijeh, «Write Perfect Code With Standard And ESLint - JSConf.Asia 2018» 9 Febrero 2018. https://www.youtube.com/watch?v=kuHfMw8j4xk [Último acceso: 19 Junio 2018].

[64] J. Lafont, «Mocky: Real HTTP mocking» https://www.mocky.io/ [Último acceso: 19 Junio 2018].

[65] Postdot Technologies, Inc., «Postman - API development environment» https://www.getpostman.com/ [Último acceso: 19 Junio 2018].

[66] Vue.js, «Vue JS Devtools - Github repository» https://github.com/vuejs/vue-devtools [Último acceso: 19 Junio 2018].

[67] Nginx Inc., «NGINX Reverse Proxy» https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/ [Último acceso: 19 Junio 2018].

Glosario

API	Application Programming Interface Interfaz de programación de aplicaciones
CDN	Content Delivery Network Red de distribución de contenidos
DOM	Document Object Model Modelo de objetos del documento
DSL	Domain Specific Language Lenguaje específico de dominio
Framework	Conjunto de módulos y estructuras ya desarrollados para ser extendido por el desarrollador y adaptado a sus necesidades.
GPU	Graphic Processing Unit Unidad de procesado de gráficos
Hypervisor	Monitor de máquinas virtuales.
JSON	JavaScript Object Notation Notación de objeto JavaScript
Kernel	Núcleo del sistema operativo. Encargado de gestionar los recursos físicos del sistema.
RAID	Redundant Array of Independent Disks Conjunto redundante de discos independientes
Scaffold	Generación automática de código a partir de unos datos o parámetros.
Snapshot	Estado exacto de una máquina en un momento dado, permitiendo su almacenamiento y posterior restauración.
SSD	Solid State Disk Disco de estado sólido
SVG	Scalable Vector Graphics Gráficos vectoriales escalables
VPS	Virtual Private Server Servidor privado virtual